IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicants:	Dempsey et al.	Confirm. No.:	3247
Serial No.:	10/627,500	Group:	2819
Filed:	July 25, 2003	Examiner:	Williams, Howard L.
For:	INTEGRATED DIGITAL CALIBRATION CIRCUIT AND DIGITAL TO ANALOG CONVERTER	Dkt No.:	AD-332J

AFFIDAVIT UNDER 37 CFR Section 1.131

We, Dennis A. Dempsey, Thomas G. O'Dwyer, Oliver J. Brennan, Alan Walsh, and Tudor Vinereanu, hereby say:

That we are the inventors for the above-identified patent application;

That we conceived in the United States the invention claimed in the above-identified patent application prior to August 30, 2002, the filing date of the cited U.S. Patent No. 6,667,703 to Reuveni et al.

Attached Exhibit A illustrates this conception of a simple and inexpensive, but much more accurate, DAC that can be achieved by integrating a calibration unit with the DAC to digitally provide the DAC transfer function end point coefficients, e.g. gain and offset coefficients, zero scale and full scale coefficients to the DAC, in which coefficients can be stored in a memory of the calibration circuit and can be applied to adjust the DAC end points.

That pursuant to this conception, we actually reduced to practice in the United States, the invention claimed in the above-identified patent application prior to August 30, 2002, the filing date of the cited Reuveni et al. patent. Attached Exhibit B illustrates a DAC that is integrated with a calibration unit to digitally provide the DAC transfer function end point coefficients to the DAC, in which coefficients can be stored in a memory of the calibration circuit and can be applied to adjust the DAC end points.

That Exhibits A and B, which relate to the aforementioned conception and actual reduction to practice, correspond to the invention broadly disclosed and claimed in the above-identified patent application.

Further deponents saith not.

Dennis A. Dempsey

Thomas G. O'Dwyer

Oliver J. Brennan

Alan Walsh

Tudor Vincreanu

Witness _____
Signature

Print Name CORMAC O SULLIVAN

Date 05/OCTOBER/2004

Witness _____
Signature

Print Name Tom TANSley

Date 05/10/2004

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicants:	Dempsey et al.	Confirm. No.:	3247
Serial No.:	10/627,500	Group:	2819
Filed:	July 25, 2003	Examiner:	Williams, Howard L.
For:	INTEGRATED DIGITAL CALIBRATION CIRCUIT AND DIGITAL TO ANALOG CONVERTER	Dkt No.:	AD-332J

AFFIDAVIT UNDER 37 CFR Section 1.131

We, Dennis A. Dempsey, Thomas G. O'Dwyer, Oliver J. Brennan, Alan Walsh, and Tudor Vinereanu, hereby say:

That we are the inventors for the above-identified patent application;

That we conceived in the United States the invention claimed in the above-identified patent application prior to August 30, 2002, the filing date of the cited U.S. Patent No. 6,667,703 to Reuveni et al.

Attached Exhibit A illustrates this conception of a simple and inexpensive, but much more accurate, DAC that can be achieved by integrating a calibration unit with the DAC to digitally provide the DAC transfer function end point coefficients, e.g. gain and offset coefficients, zero scale and full scale coefficients to the DAC, in which coefficients can be stored in a memory of the calibration circuit and can be applied to adjust the DAC end points.

That pursuant to this conception, we actually reduced to practice in the United States, the invention claimed in the above-identified patent application prior to August 30, 2002, the filing date of the cited Reuveni et al. patent. Attached Exhibit B illustrates a DAC that is integrated with a calibration unit to digitally provide the DAC transfer function end point coefficients to the DAC, in which coefficients can be stored in a memory of the calibration circuit and can be applied to adjust the DAC end points.

That Exhibits A and B, which relate to the aforementioned conception and actual reduction to practice, correspond to the invention broadly disclosed and claimed in the above-identified patent application.

Further deponents saith not.

Dennis A. Dempsey

Thomas G. O'Dwyer

Alan Walsh

Oliver J. Brennan

Alan Walsh

Tudor Vinereanu

Witness Duane W. Younkin
Signature

Print Name Duane Younkin

Date 10/05/2004

Witness David Jon Basson
Signature

Print Name DAVID JON BASSON

Date 10/05/2004

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicants:	Dempsey et al.	Confirm. No.:	3247
Serial No.:	10/627,500	Group:	2819
Filed:	July 25, 2003	Examiner:	Williams, Howard L.
For:	INTEGRATED DIGITAL CALIBRATION CIRCUIT AND DIGITAL TO ANALOG CONVERTER	Dkt No.:	AD-332J

AFFIDAVIT UNDER 37 CFR Section 1.131

We, Dennis A. Dempsey, Thomas G. O'Dwyer, Oliver J. Brennan, Alan Walsh, and Tudor V nereanu, hereby say:

That ve are the inventors for the above-identified patent application;

That ve conceived in the United States the invention claimed in the above-identified patent application prior to August 30, 2002, the filing date of the cited U.S. Patent No. 6,567,703 to Reuveni et al.

Attached Exhibit A illustrates this conception of a simple and inexpensive, but much more accurate, DAC that can be achieved by integrating a calibration unit with the DAC to digitally provide the DAC transfer function end point coefficients, e.g. gain and offset coefficients, zero scale and full scale coefficients to the DAC, in which coefficients can be stored in a memory of the calibration circuit and can be applied to adjust the DAC end points.

That pursuant to this conception, we actually reduced to practice in the United States, the invention claimed in the above-identified patent application prior to August 30, 2002, the filing date of the cited Reuveni et al. patent. Attached Exhibit B illustrates a DAC that is integrated with a calibration unit to digitally provide the DAC transfer function end point coefficients to the DAC, in which coefficients can be stored in a memory of the calibration circuit and can be applied to adjust the DAC end points.

That Exhibits A and B, which relate to the aforementioned conception and actual reduction to practice, correspond to the invention broadly disclosed and claimed in the above-identified patent application.

Further deponents saith not.

Dennis Dempsey
Dennis A. Dempsey

Thomas G. O'Dwyer

Oliver J. Brennan

Alan Walsh

Tudor Vinerean

Witness Ann O'Brien
Signature

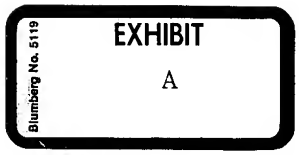
Print Name ANN O'BRIEN

Date 4/10/04

Witness Michelle Breen
Signature

Print Name MICHELLE BREEN

Date 4/10/'04



Appendix No. 1:
Engineering Notebook No. 4761, pp.11-14
Dennis A. Dempsey
October 10, 1997.

0 Oct '47 DAC Calibration

To document technical conversation / discussion / development with Tom O Dwyer on Sept '29 97.

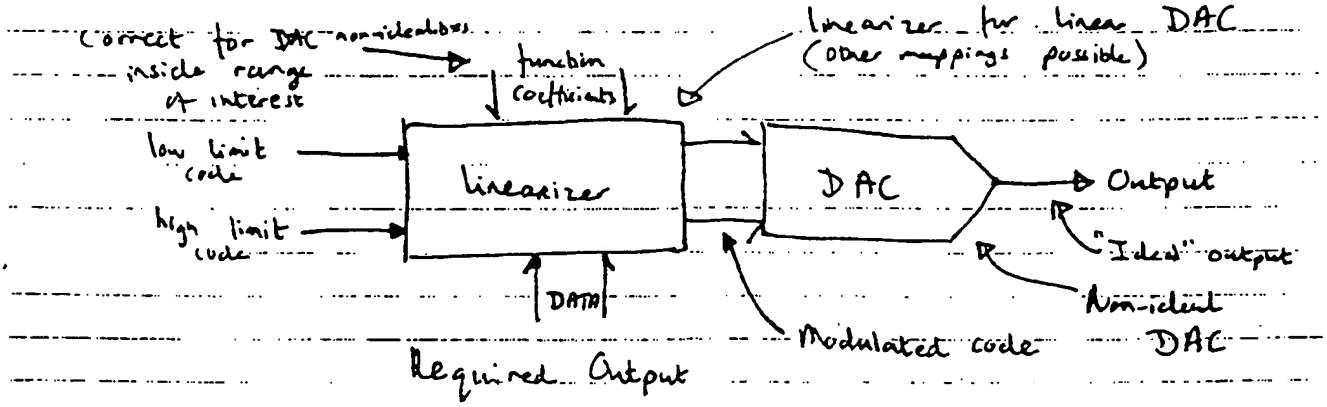
Point Calibration has been used for many years, an earlier form being laser trimming and another being digital look-up.

To idealize, normally linearize, between two points a mapping function must be used. This function has many forms of possible implementation some more difficult (complex and/or large) than others and hence have different levels of usefulness in different applications.

Radamacher & Walsh functions are the best known digital functions for this purpose and there digital bit-wise definition lends themselves naturally to easier digital implementation and easier understanding. → Hence their common use & teaching at under-graduate level.

The constraint they have is that are should / has to use 2^N points to for this purpose as the natural period for Radamacher functions is 2^N.

Other functions can be used alternatively, but the concept remains the same.



DATA may "zoom in" on a section (modulating "low limit" & "high limit") and also the function coefficients for that section.

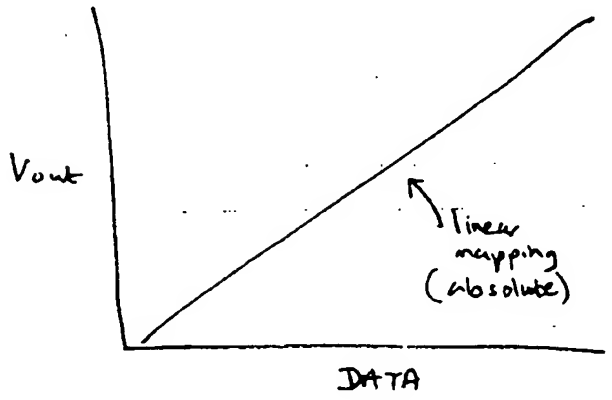
Tom O'Dwyer 23rd Feb 2001

Noel Geraghty 5/Oct/98 Donncha Demosen

10 Oct, 97 DAC Calibration

Work on same vein continued since ~~Oct 3rd~~ Sept 29th.

High-level Requirent (linear case) :



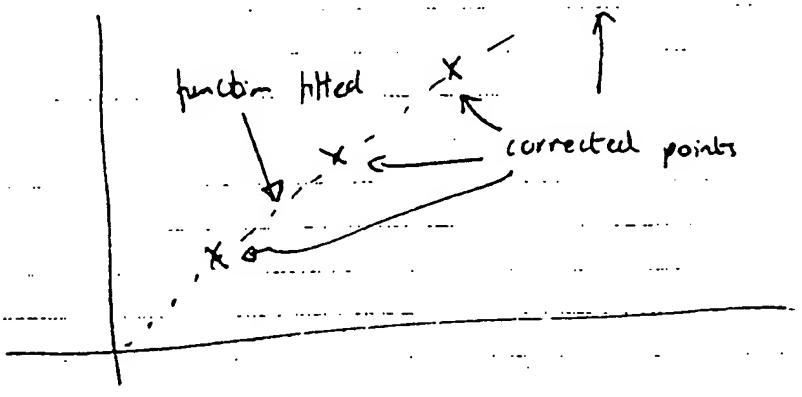
Non-ideal correction

(1) Correct all / many points (all the extremity)
→ "some" points allows interpolation or function fitting between these points

(2) Fit Non-ideal T.F. of DAC to linear via function mapping.

(3) Use an amalgam of (1) & (2). (Probably best)

Using (1) to do partial absolute correction on some points



Use function fitting to do the rest.

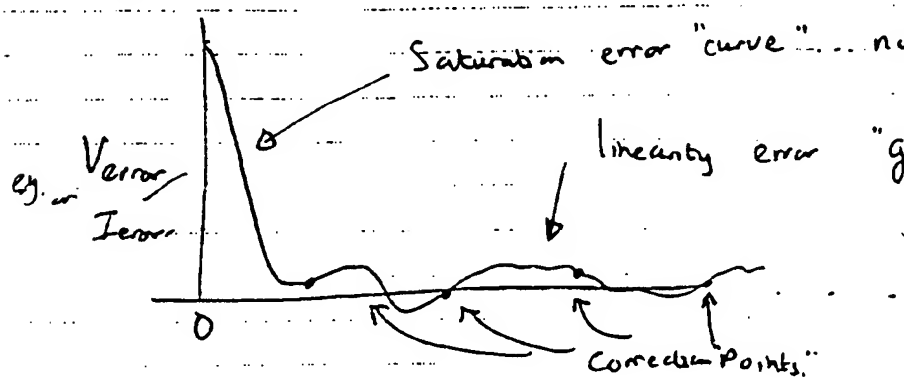
23rd Feb 2001

Analog Circuits Lecture — Dennis Demopoulos

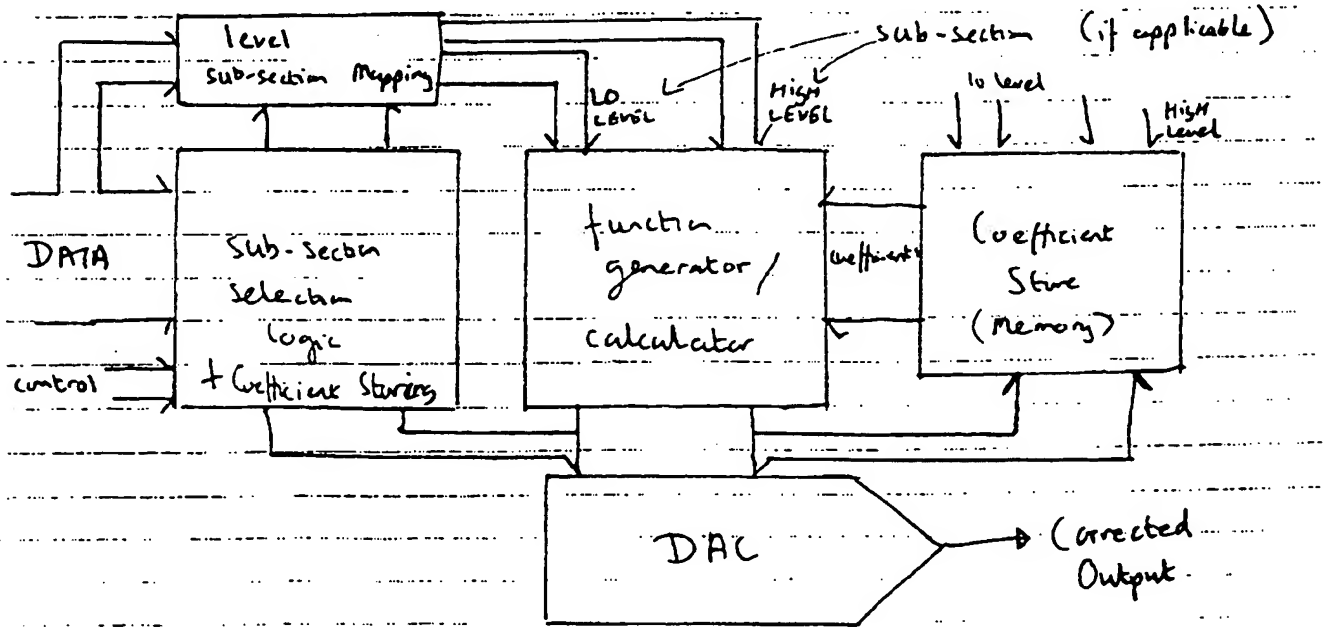
16 Oct '91

DAC Calibration.

In applications where the DAC signal path has "unusual" code dependency changes then one function generator may be non-optimal eg. rail operations where amplifiers go out of saturation. In this case the function may require to be changed, and a new function more suitable to the characteristic in this region could be used.



For this purpose the correction "points" may be deliberately movable to get better end error (finite correction due to practical realization trade-offs) by isolating the different areas from each other more exactly.



Donal Geraghty 5/Oct/98 Dennis Dempsey

Dennis Dempsey 23rd Sept 2001

-24 Oct

DAC Paket Prep. (p1)

Actions: Prepare functional analysis of DAC architecture for R_1, R_2 & R_3 — for ① Ideal (Rsw = 0, Matching / Mismatch perfect)
② Non-ideal

$\rightarrow f(R_1, R_2, R_3, R_{SW})$

→ Check Truth table functionality.

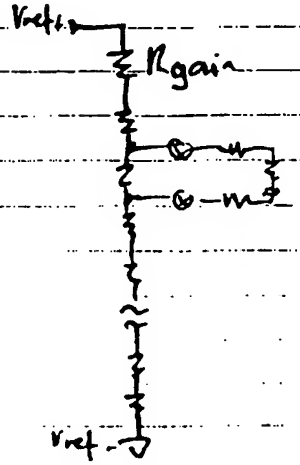
Dennis Donna

Appendix No. 2:
Engineering Notebook No. 4761, pp.24-27
Dennis A. Dempsey
April 1-2 1999

April '99

Calibrated DACs

The AD5300 calibration has been debated in ADI for quite some time. Here are more of the thoughts I have on this subject (have had for quite some time)



ADSICO Style DAC Configuration Diagram

Main-Dac / Sub-DAC DACs are a trait common to almost all linear DACs made in such a fashion which means the two are essentially seperate.

① Static Calibration
② Dynamic Calibration
③ Background Calibration

are three groupings used to cover many of the techniques used to correct for these non-idealities.

ΣΔ techniques fall under the "dynamic calibration" title as DSP is used to modulate non-idealities in a preferred manner.

Dynamic Element Matching can be used as one technique in ΣΔ designs but is essentially a seperate dynamic technique, not specifically focussed on the integral action associated with ΣΔ.

Background calibration can be grouped with either/or static or dynamic (normally dynamic) as it usually involves switched elements such that non-idealities are "modulated", when the converter is quiesc-

DATE 23rd Feb 2001 SIGNED Dennis Dempsey

1 April '99 Calibrated DACs

is not using the particular element(s). Hence the reference to "background" as this technique undergoes this calibration while the converter is in use e/and "busy" in such a manner not to disturb normal operation.

Static Calibration can be permanent (eg laser-burn) pseudo-permanent eg EEPROM or held in memory while powered eg standard CMOS logic. This is a "once-shot" approach where the converter is calibrated at a particular point in time & during or post-production, and this holds this calibration "memory" (analog or digital) for continued usage eg AD7853 ADC. Statically calibrated converted designs have (unlike dynamic & background as much) the difficult constraint of maintaining their calibrated performance level and rejecting drift mechanism such as temperature change, supply change, reference change & device drift.

SEEN AND UNDERSTOOD

DATE 23rd Feb 2001 SIGNED Dennis Dempsey

2 Apr '99 Cambrated DACs

systems. Digitized Basis functions "seem" more naturally suited to digitized systems. Hence the earlier mention of Radamacher and Walsh functions. Simply binary is a basis function also!! It is notable that the dominant linearity variability mechanisms may be more suited to a particular basis function set.

Implementation of same, in practice, is the other key issue. Different design topologies and methodologies will be used for different implementations. Economic and engineering balance is called for in order to then complete the optimal solution.

Piece-Wise-Linear (P.W.L) calibration or bilinear interpolation or point corrected calibration are three titles for essentially the same methodology which has been often used in electronic circuits and systems due to its simple but effective nature.

Bilinear $\Sigma\Delta$ is a favoured technology for reasons that it uses up the full domain very well (in frequency terms) and it lends itself to implementation very well

PWL is numeric friendly as it has ① points/values ② subsection basis function coefficients. Because the sections' size can be chosen (design parameter) and the designer is free to use whatever basis function he/she prefers in same (second design parameter) s/he can use varying code resolution (third design parameter) this leaves scope for different favoured approaches depending on the performance level(s) required and the designer's(1) effect on optimization. This is a high level challenge based on the scope of errors, and their nature, to be catered for in the calibration. Synthesis is useful for optimization, over design parameter constraints, of many of the functions involved.

Dennis Dempsey

23rd Feb 2001

Appendix No. 3
Engineering Notebook No. 5849, pp.1-8
Oliver J. Brennan
March 14, 2001.

Note: Diagram on p8, signed & dated 16/2/'00

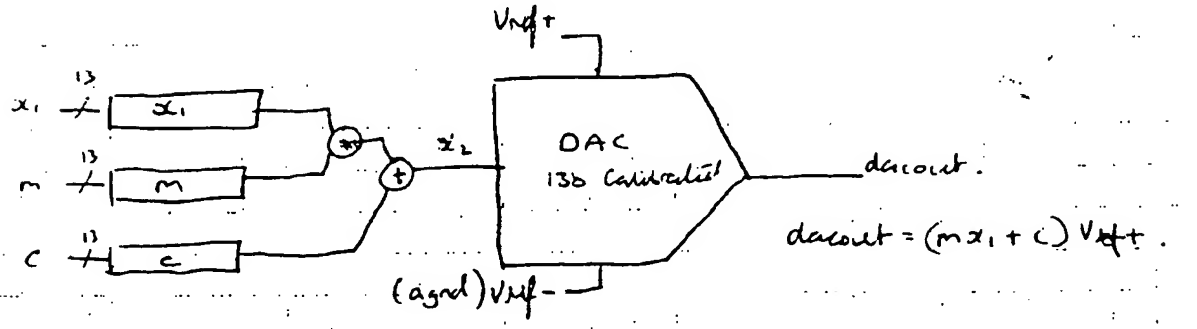
May 14, 2001 Calibrated DAC.

The AD5379 Calibrated DAC (see Denis-A Dempseys notebook details the calibration method for this structure) allows a user to do a global gain and offset calibration by writing gain (m) and offset (c) co-efficients to gain and offset registers.

A digital word written to a dac is then digitally manipulated as per Eqn 1.

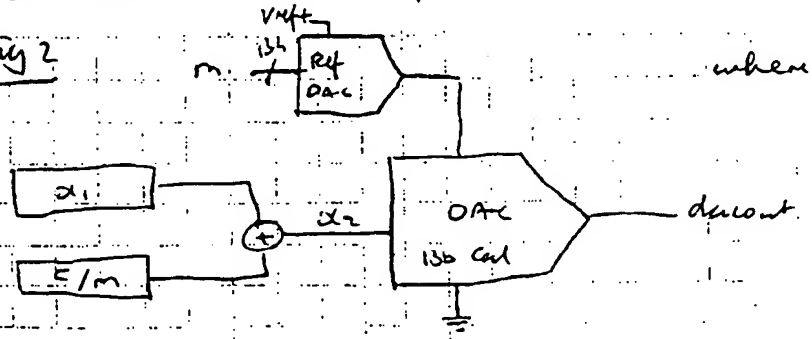
$$x_2 = m x_1 + c \qquad \text{Eqn 1}$$

where x_1 is digital input word (13 bit in AD5379) and m and c are 13b gain and offset register contents.



The m, gain adjust could be done in the analog domain by using another DAC to drive the Vref+ input. This "reference dac" would have m as a digital input. A similar 13b cal dac could be used here. One implementation is shown in Fig 2

Fig 2



when $x_2 = x_1 + \frac{c}{m}$

$dacout = (mx_1 + c) V_{ref+}$

The Advantage of this scheme is the DAC transfer function can be corrected by m without introducing dnl errors which occur in digital gain, offset adjust.

Denis Dempsey 14/3/02 Oliver Brennan May 14, 2001

Nov 30, 2001 Single Supply Calibrated DAC

To achieve rail-to-rail Dac Transfer function standard practice is to over range the output span so some calibration routine can be implemented to adjust the transfer function to match the ideal desired one.

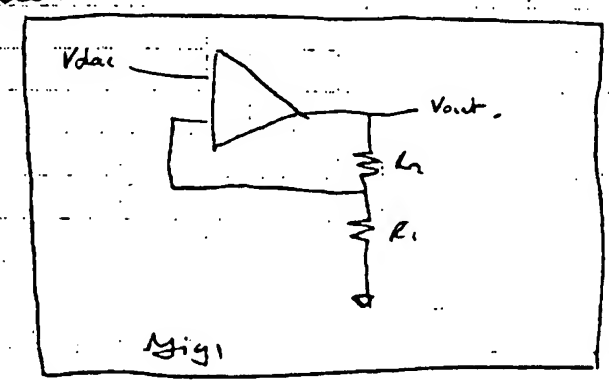
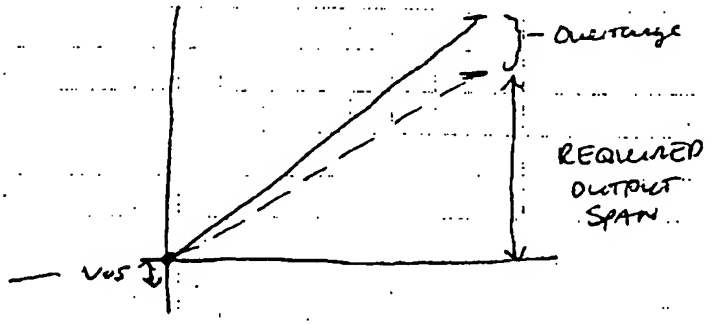


Fig 1

In Single Supply implementation the buffer amplifier offset voltage should be designed to always be negative. This will ensure that there never is a positive zero code Error. The closed loop gain has to be overranged by, at least this amount to allow the desired maximum output span to be achieved

Fig 2



Amp Vos forced to be -ve so will hit -ve rail to output Readout capability

Dennis Dempsey 19/3/02 Dennis Dempsey Nov 30, 20

Dec 3, 2001 "1 xlog Clear function"

In a Voltage DAC where the Analog Output Voltage is related to the digital input code and a positive and negative reference voltage this Output voltage can be switched to some pre-selected clear voltage (e.g. gnd) as follows:

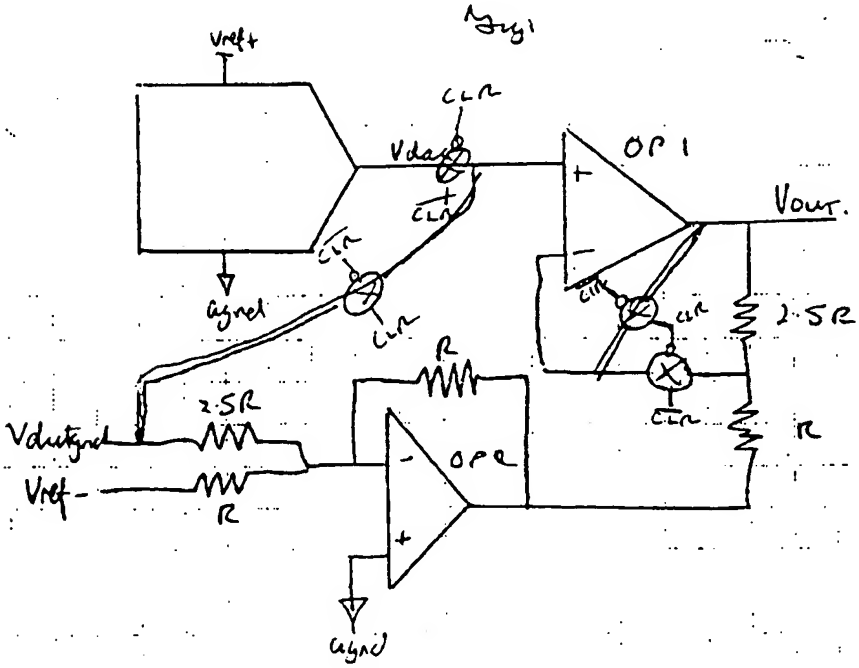
E.g. AD5379 DAC Channel

where $V_{out} = 3.5\ V_{dac} + 2.5\ V_{ref-} + V_{dutgnd}$ Eqn 1

where $V_{dac} = 0$ to V_{ref+} Eqn 2

This configuration is shown in fig 1. [Note $A_{gnd} = 0$ in above Eq

Fig 1



In Fig 1 the Signal paths detailed in blue font show a switching network which re-configures the circuit from its normal operation (as per Eqn 1) to clear mode where the Vdac input of op amp 1 (OP1) is tri-state and Vdutgnd is switched into OP1. OP1 is re-configured as a unity gain buffer. Thus $V_{out} = V_{dutgnd}$ ..... where V_{dutgnd} is the required "Clear Voltage".

Denis Dempsey 19/3/02 Aiden Bonar Dec 3, 2001

Dec 3, 2001 Overranging to Enable TUE Calibration

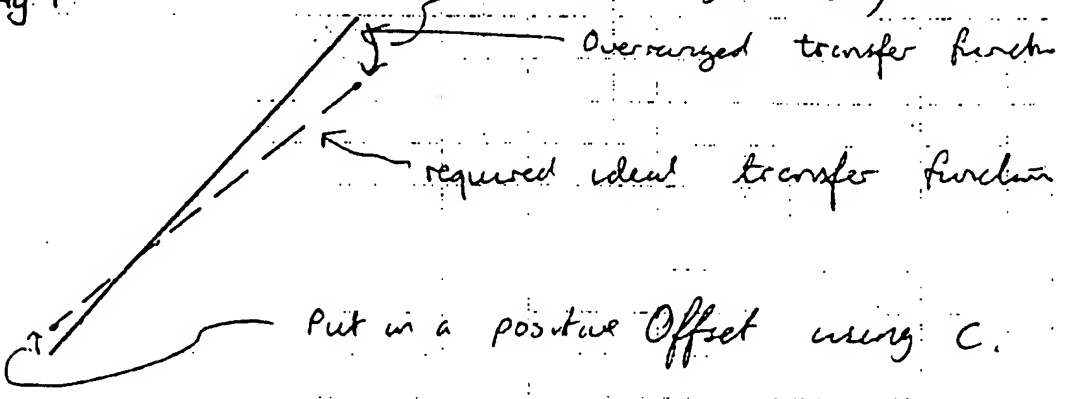
Page 2 discussed Overranging with respect to a Single Supply DAC.

In a dual Supply DAC, as per AD5379 Product, it is more straight forward to be able to cope with zero-scale and full-scale Errors.

The digital calibration of the String DAC core in the AD5379 digitally adjusts the transfer function to illiminate offset Error, Gain Error and linearity errors in the transfer function. i.e Total Un-adjusted Error --- to 13b accuracy.

This calibration principle can use a 'gain, m and Offset, c, register to turn down the gain and provide a d.c. offset (+ or -) to illiminate offset Error. However for this to work the actual output range of the dac channel has to be deliberately 'Over ranged'. It should be over-ranged by a sufficient amount to allow enough range to "cater" for all offset + Gain Errors inherent in the architecture as it is manufactured

Fig 1



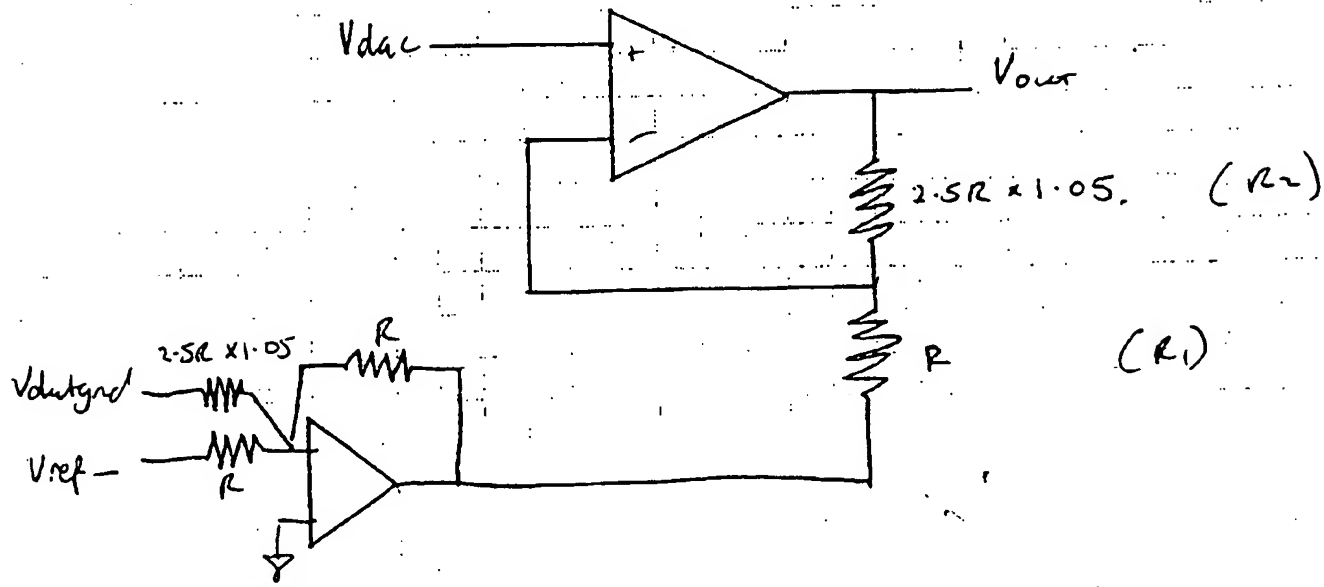
To achieve this overranging the DAC's output buffer configuration shown in Fig 2 on next page can be altered to increase gain by 5% (in this example)

Denis Dempsey 19/3/'02

Oliver Brennan Dec 3, 2001

Dec 3,2001 Over ranging to ensure the ...

Fig 2



Nominally : $V_{out} = 3.5\ V_{dac} + 2.5\ V_{ref-} + V_{dutgnd}$

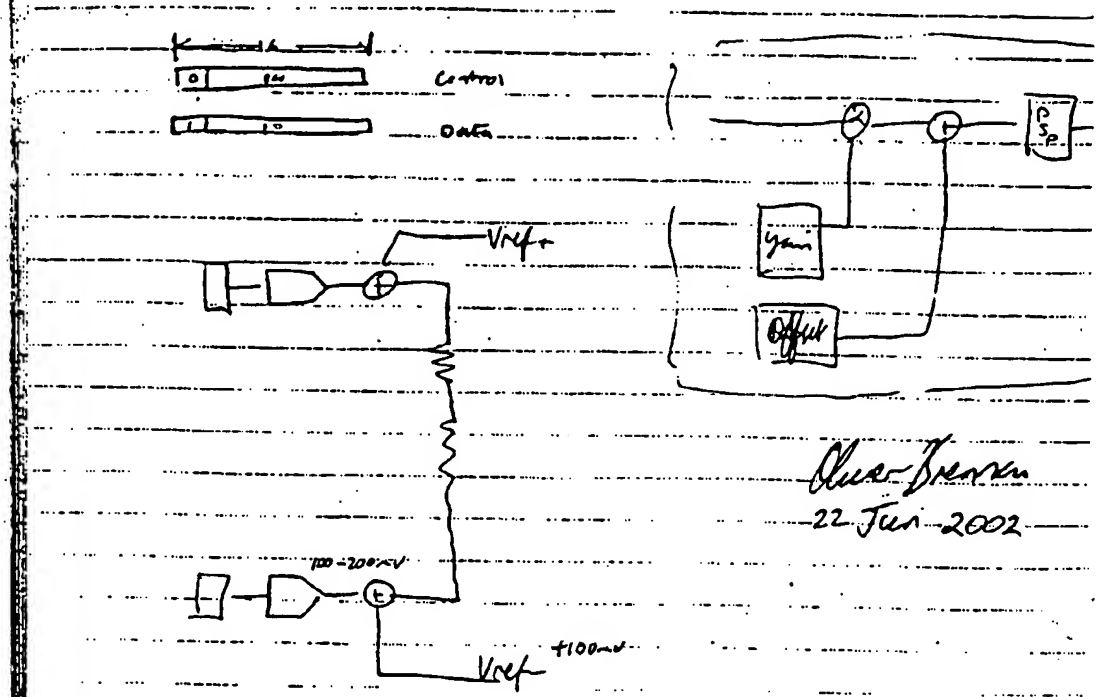
Overranged : $V_{out} = 3.5{+}5\%\ V_{dac} + 2.5 + 5\%\ V_{ref-} + V_{dutg}$

In the above Example a 5% overranging is chosen, This the assumes that the sum of all offsets in the channel [Amp offsets, I.R in Gnd line etc.] and all the Gain Error [R2, R1 etc Gain Resistors) ~~to~~ are less than 5% of fullscale Output Voltage

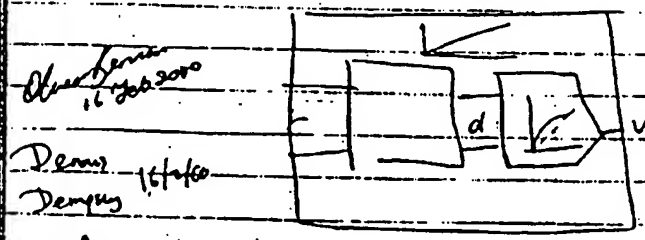
Denis Dempsey 19/3/'02

Oliver Brennan
Dec 3, 2001

Jan 22, 2002 Digital Calibration Method for DAC

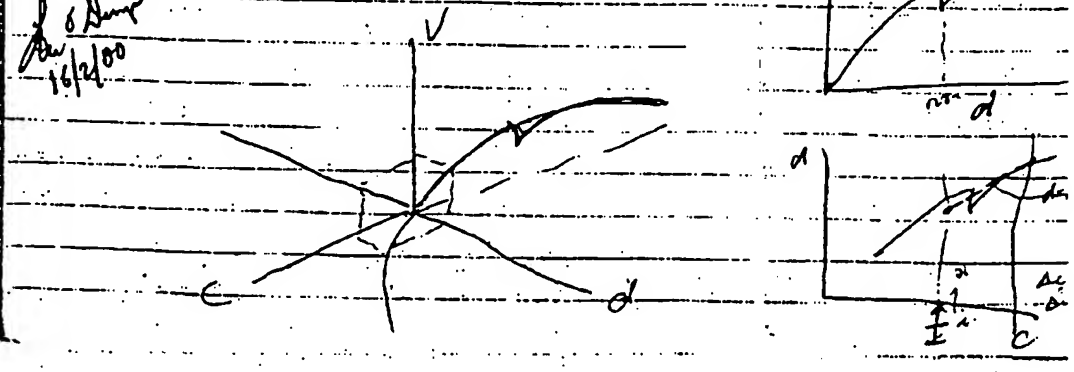


Oliver Brennan
22 Jan 2002



Oliver Brennan 16 Jan 2002

Dennis Dempsey 16/1/02



Dennis Dempsey DATE 16/1/02 SIGNED Oliver Brennan DATE Jan 22, 2002

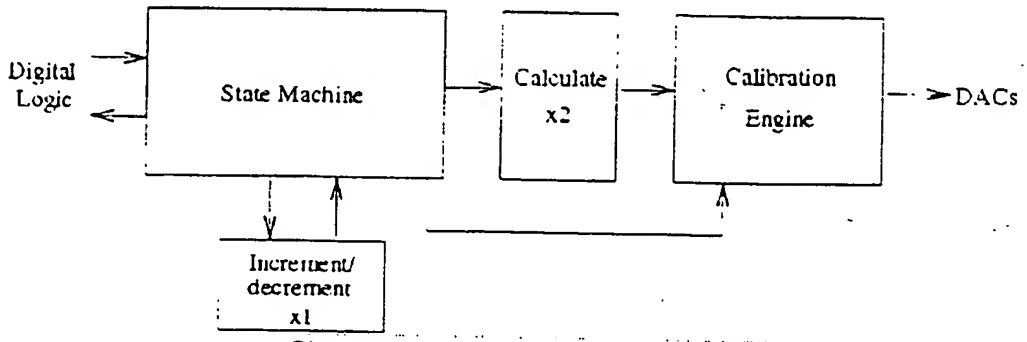
Appendix No. 6
Section 3.9 of the AD5379 Design Document of Tudor Vinereanu.

This section also contains information not relevant to the submission, as multiple functions are amalgamated in the design, as part of the optimization.

3.9. State Machine and Calibration Engine

The state machine and the calibration engine are fully synchronous blocks, running with the same clock. They are treated separately during synthesis and place&route, due to need of tighter control over the timing, testability issues (scan test) and clock tree insertion.

A block diagram is shown below.



Block diagram of state machine + calibration engine

The state machine is used to do the following main tasks:

- initialize the part after power-up, after a RESET command or a hard RESET
- command decoding in user mode
- command decoding in test mode, including the test commands
- control the increment/decrement function for x1
- control the calculation of x2
- start the calibration engine and schedule the codes for calibration

The calibration engine receives the x2 code as an input, calibrates it and outputs the x3, which is then written to the respective DAC.

The other two blocks in the figure are implementing aritlmetic functions.

3.9.1. Increment/Decrement x1

This blocks implements the increment/decrement function by using an adder. The operation is:

x1_incdec[13:0] = x1[13:0] + offset[7:0]

x1 and *x1_incdec* are unsigned numbers, and *offset* is signed represented in 2's complement.

A DesignWare adder is instantiated for the operation:

```
wire [13:0] x1_incdec;
wire x1_incdec_sign;

adder1 adder1(.inst_A({1'b0, x1_reg}), .inst_B({{7{offset_reg[7]}}, offset_reg}), .SUM_inst({x1_incdec_sign,
x1_incdec}));
```

Underflow/overflow are not allowed for the result, so 2 flags are used to detect these exceptions:

```
wire x1_uf;
wire x1_of;

assign x1_uf = x1_incdec_sign & offset_reg[7];
assign x1_of = x1_incdec_sign & (~offset_reg[7]);
```

x1 clamps a zero scale/full scale in case of an underflow or overflow respectively.

3.9.2. State Machine

This block is a Mealy state machine with 81 states binary coded using a 10-bit state register. Even though the 81 states could be coded using only 7 bits, the idea was to use a pseudo-one hot state encoding to speed up state decoding logic, which was a speed bottleneck. It is probable that now there is no need for this anymore, but it was easier to leave the state encodings as they are, with a possible overhead of 3 flip-flops.

Due to the complexity of the state machine, it will not be detailed here. Only the main issues will be presented.

3.9.2.1. Initialization Sequence

The state machine executes the initialization sequence in any of the following situations:

- after power-up
- after a hard RESET
- after a soft RESET command

The sequence of operations during initialization is the following:

1. - read default value for m register from EEPROM (2 MSBs ignored in the EEPROM word)
   -> 1 EEPROM read
2. - read default value for c register from EEPROM (2 MSBs ignored in the EEPROM word)
   -> 1 EEPROM read
3. - read default value for x1 register from EEPROM (2 MSBs ignored in the EEPROM word)
   -> 1 EEPROM read
4. - write the FIFO SRAM (128 words) with the pattern 000000 (increment with step 0, is equivalent with 'no operation'). This is in case the read pointer in the FIFO doesn't advance in time due to synchonization issues between the clock domains.
   -> 128 writes to FIFO SRAM port 1
   - write the default m to locations 0-63 in the user SRAM
   -> 64 writes to user SRAM port 1
   - write the default c to locations 64-127 in the user SRAM
   -> 64 writes to user SRAM port 1

5. - write the default x1 to locations 128-167 in the user SRAM, calibrate the initial codes for the DACs and write them to the DACs
   -> 40 writes to user SRAM port 1
   -> 40 reads from user SRAM port 1
   -> 40 reads from user SRAM port 2
   -> 80 reads from the EEPROM
6. - read EEPROM options and update relevant registers
   -> 2 EEPROM reads

The clock trimming bits are double buffered, so the signals going to the clock generator are updated only after the initialization sequence finishes (the clock generator is disabled)

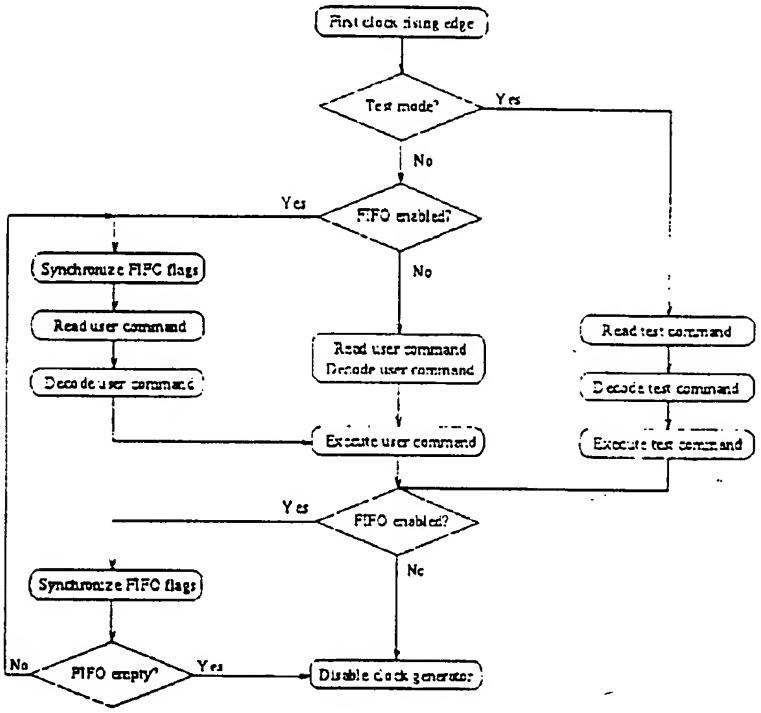
During initialization, the EEPROM options have their default values (specified in the EEPROM options section). They are only modified in the end of the initialization sequence (step 6).

The number of clock cycles required for an EEPROM read (<eo_read_clk_count>) is read in the second EEPROM read of step 6. Because no other EEPROM read occurs afterwards (before the initialization sequence finishes), reading a random value after the first power-up should not affect the EEPROM read counter.

3.9.2.2. State Diagram

A simplified state diagram is shown in the figure below. The state machine makes distinction between test mode and user mode, decoding and execution of test commands being separate. The figure is slightly incorrect with respect to the test modes: it is possible to use FIFO in test mode, but this was not shown to keep the figure simple.

Also, there is a difference between FIFO enabled mode and FIFO disabled mode. If the FIFO is enabled, first step is to synchronize FIFO flags (*pop_empty*), then continue with decoding and execution. If FIFO is disabled, there is no need for synchronization, but also, reading and decoding the command are performed in the same step. Practically, this is done before the first rising edge of the clock, while the state machine is waiting, and when the edge comes, it jumps directly to the first execution state.

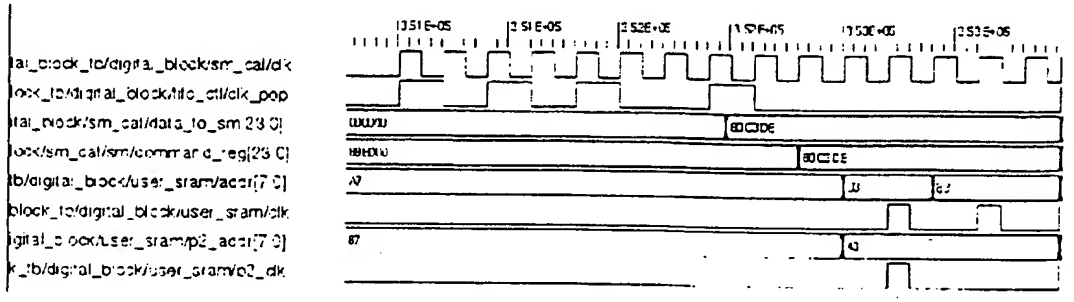


State diagram

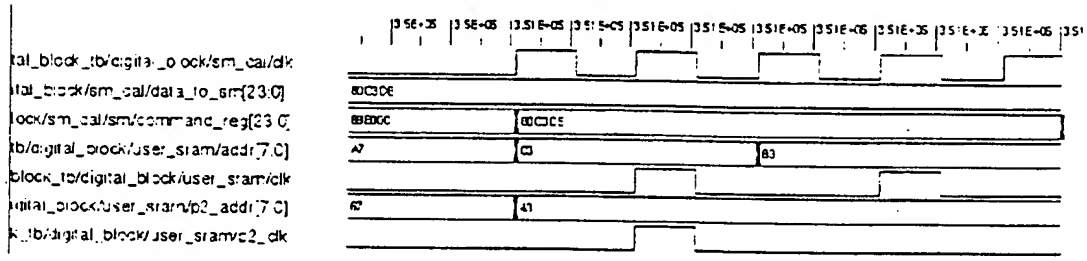
The reason to do this was to minimize as much as possible the execution time for a command. Working with the FIFO enabled means there is a big overhead due to synchronization between clock domains, so it is only efficient if the user writes a number of commands at full interface speed. Otherwise, if the preferred way is to write one command at a time, execution time for a single command is emphasized.

The next figure provides an example of how this works. The top waveform shows the case when the FIFO is enable. First, 3 *clk_pop* pulses are generated to synchronize the FIFO, then a fourth pulse reads the command from the SRAM. *command_reg* is updated with the new command, which is decoded during the next clock cycle. This causes the user SRAM addresses to be calculated (*user_sram_addr* and *user_sram_p2_addr*), which represents the first step in the execution of the command.

The bottom figure shows the case when the FIFO is disabled. *command_reg* and user SRAM addresses are updated at the same time, on the first rising edge of the clock. This shows that the execution of the command starts straight away from the first clock pulse.



Command decoding with FIFO enabled



Command decoding with FIFO disabled

3.9.2.3. User Command Execution

The soft RESET command resets the logic by starting the state machine from the RESET state (synchronous reset) and generating an asynchronous reset signal for the rest of digital logic. The clock generator is enabled all the time, since the state machine is working.

The write commands to the registers and increment/decrement function have three options, as seen in the datasheet:

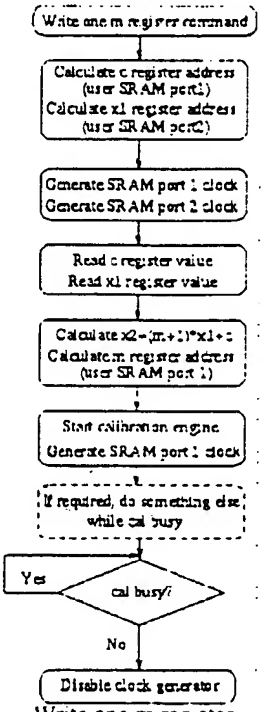
- write/increment/decrement one channel
- write/increment/decrement multiple channels (2, 3 or 4)
- write/increment/decrement all 40 channels

Increment/decrement applies only to *x1* registers, while the write commands apply to *m*, *c* or *x1*, according to the status of REG1 and REG0.

All the commands require the resulting code to be calibrated.

All these commands are implemented based on the write command for one channel. Therefore, this command will be detailed first, and then it will be shown how the others call this command to execute. All the explanations and examples will be given for the situation when FIFO is disabled, due to simplicity. If the FIFO is enabled, the execution stays the same, only the overhead for flag synchronization is added.

The state diagram for writing one *m* register is shown. The first thing to do is the read the values of the other 2 registers (*c* and *x1*) needed to calculate *x2*. Their addresses are calculated in parallel and they are read in parallel from the user SRAM (port 1 and port 2). *x2* is calculated during 2 clock cycles (as explained later in section 3.8.5), and while this is done, the value of *x1* is written to the user SRAM. Then the value of *x2* is passed to the calibration engine and the state machine waits until the calibration is complete to disable the clock generator and go back to sleep mode.



Write one m register

After *x2* has been passed to the calibration engine, the state machine may do something else. This is not relevant with respect to the command to write one register, but makes sense in case a write multiple/ all 40 channels command is executed or if the FIFO is enabled. In the latter situation, the state machine may read, decode and start the execution of the next command in the FIFO, or synchronize the flags.

Writing one *c* or *x1* register is similar:

- write one *c* register:
  - read *m* register from user SRAM port 1
  - read *x1* register from user SRAM port 2
  - write *c* register to user SRAM port 1
- write one *x1* register:
  - read *m* register from user SRAM port 1
  - read *c* register from user SRAM port 2
  - write *x1* register to user SRAM port 1

The steps taken to execute a "write one channel" command are enumerated below (each step takes one clock cycle):

****** Calculation of x2 = (m + 1) * x1 + c ******

1. - read command from the interface
   - calculate addresses in user SRAM. In the next step, the remaining 2 operands needed for calculating x2 will be fetched from memory.
2. - apply a clock pulse on both ports of user SRAM
3-4. - read the values from both SRAM ports
   - perform the operation x2 = (m + 1) * x1 + c
   - write to user SRAM as requested by the original command

****** Calculation of DAC code (x3) ******

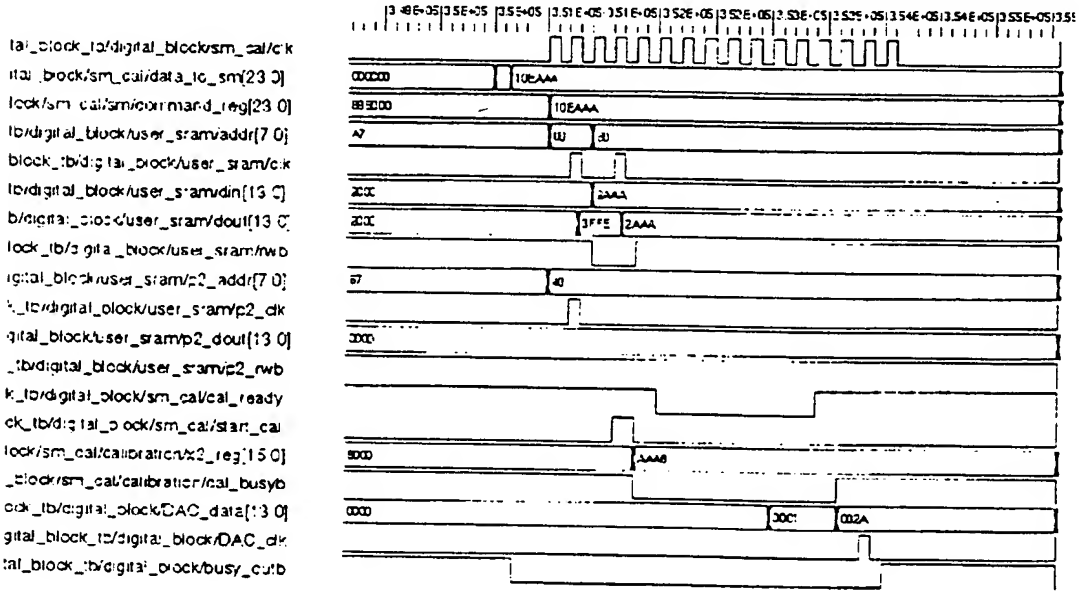
5-8. - read EEPROM word 0 (assuming 4 clock cycles for EEPROM read)
9-12. - read EEPROM word 1
   - perform partial calculations
13-14. - finish calculating x3
15. - generate DAC clock pulse

A special mention has to be made about the *c* register. Its value is a signed number, which could be written to the interface in Advantest format or 2's complement, depending on the status of

eo_offset_format, but internally it is stored as a 2's complement number. If *eo_offset_format*=1, the format is 2's complement, so the value is written in the user SRAM as it is. However, is *eo_offset_format*=0, the format if Advantest format, and the number is converted to 2's complement prior to writing to the SRAM. For this, only the sign bit has to be inverted:

```
c_reg <= {command_reg[13] ~^ eo_offset_format, command_reg[12:0]};
```

The example below shows the execution of command A7..0=1000000, REG1=1, REG0=1, DB13..0=10101010101010, which means write 2AAA to the *x1* register, DAC 0. First, the *m* and *c* registers values are read from the dual-port SRAM (from addresses 0x00 and 0x40 respectively). Then, while *x2* is calculated, the *x1* register is written in the SRAM (address 0x80, port 1). *user_sram_rwb* is low during the write. When *x2* is available, the state machine checks the status of *cal_ready*. *cal_ready*=1, so the calibration engine is waiting for a new *x2* code. *start_cal*=1 showing that the *x2* is valid and can be read, which causes the value of *x2* to be read and subsequently *cal_ready* goes low and *cal_busyb* goes high. The meaning of these signals will be explained in section 3.8.5. The state machine waits until *cal_busyb* goes high again to finish the execution by turning off the clock.

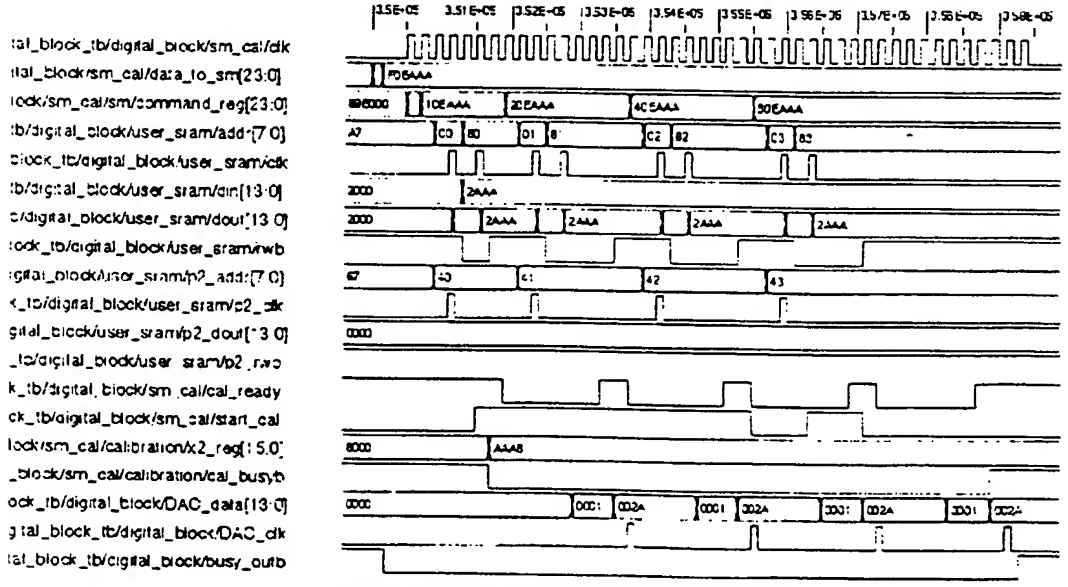


Write x1 register (one channel) without FIFO

As previously stated, write multiple/all 40 channels commands use the "write one channel" command during execution. The multiple channel commands use the address lines A7..A4 to specify the groups to which they apply. The sequence will be shown for a command with A7..A4=1111, all the others being easy to derive from that.

- read write one register command with A7..A4=1111
- store_groups = A7..A4 (save for future reference)
- write A7..A4 = 0001 to command_reg, so a "write one register from group 1" command will be executed
- inspect store_groups to see if the execution is finished -> it's not
- write A7..A4 = 0010 to command_reg, so a "write one register from group 2" command will be executed
- inspect store_groups to see if the execution is finished -> it's not
- write A7..A4 = 0100 to command_reg, so a "write one register from group 3" command will be executed
- inspect store_groups to see if the execution is finished -> it's not
- write A7..A4 = 1000 to command_reg, so a "write one register from group 4" command will be executed
- inspect store_groups to see if the execution is finished -> it is
- finish execution

An example is provided in the figure below. It is easy to see how the original command, with A7..A4=0xF, is executed as 4 distinct commands with A7..A4=0x1, 0x2, 0x4 and 0x8 respectively. Also, the time to change *command_reg* from the first command to the second is less than the time required to change to the third or fourth command. This is because the state machine passes the first *x2* (from the command 0x10EAAA) to the calibration engine and jumps to executing the next command 0x20EAAA. When the second *x2* is ready, it check the status of cal_ready, but this is low, so it has to wait until *cal_ready*=1.



Write x1 register, all four groups, without FIFO

If the command requires to write all 40 channels, then a similar approach as with multiple channels was adopted. This time, the "write all 40 channels" command is split into 10 "write 4 groups" commands, with A3..0 ranging from 0000 to 1001.

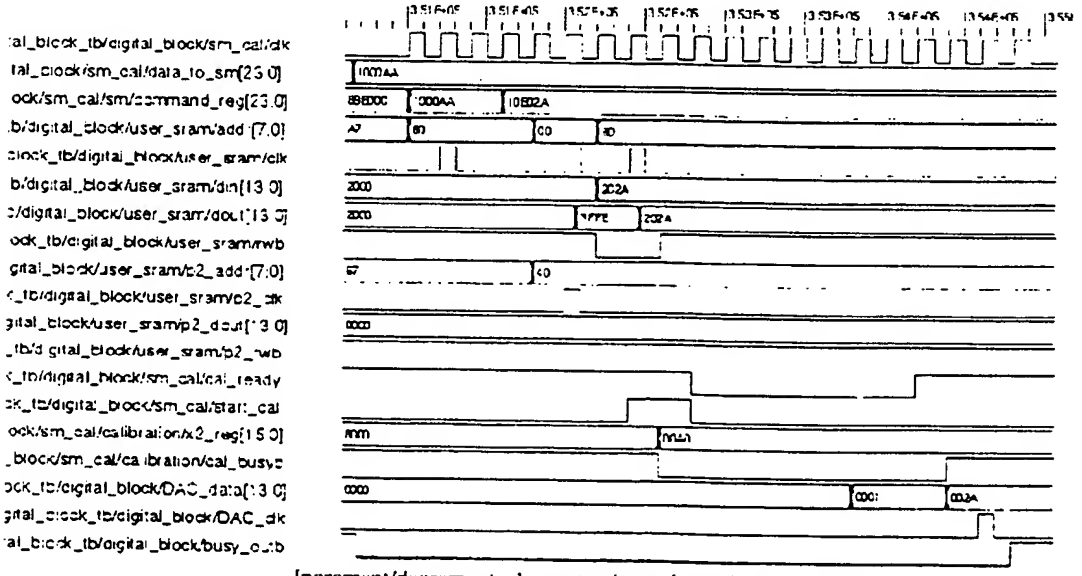
An increment/decrement command is executed following these steps:

- read *x1* from user SRAM
- increment/decrement *x1*
- execute a "write one *x1* register command", with the value of *x1* being the incremented/decremented one

The same comment applies for the increment/decrement step as for the the *c* register: it may be written in Advantest format, but internally it has to be represent as a 2's complement number:

```
if (command_reg[8])
  offset_reg <= {1'b0, command_reg[6:1], 1'b0};
else begin
  offset_reg[7:1] <= ~({command_reg[8], command_reg[6:1]}) + 1;
  offset_reg[0] <= 1'b0;
end
```

This is exemplified in the figure for an increment command. The first 4 clock cycles are dedicated to read the value of *x1* from user SRAM (port 1), do the increment required, then assemble the "write one x1 register" command and write it to *command_reg*. The addresses for *m* and *c* registers in user SRAM are ready after the 5th clock cycles, and from then on the same steps as for writing the *x1* with the incremented value (0x202A as opposed to the original 0x2000).



Increment/decrement x1 register (one channel) without FIFO

Some evaluation results regarding execution times for commands are presented in the following table. The timing is expressed in clock cycles and represents the number of clocks required for the BUSY signal to go high (inactive) again. However, in a real situation, there are a number of other delays how contribute to the BUSY low time: delays through the input pad ring, delays

through internal logic, powerup delay for the clock generator, output delay through the BUSY pad.

An assumption was made that 4 clocks are needed per EEPROM read, which is the figure expected based on experiments and an internal clock speed of 20ns. This parameter affects the total execution time by adding roughly 2 clocks per command for each additional clock cycle needed to read the EEPROM.

Command	Execution time [clocks]	
	FIFO enabled	FIFO disable
write one channel	33	15
write two channels	44	26
write three channels	53	35
write four channels	62	44
write all 40 channels	388	370
increment/decrement one channel	37	19
increment/decrement two channels	57	39
increment/decrement three channels	75	57
increment/decrement four channels	93	75
increment/decrement all 40 channels	743	725
10 * write one channel	114	150*
50 * write one channel	474	750*
100 * write one channel	924	1500*
10 * increment/decrement one channel	226	190*
50 * increment/decrement one channel	1066	950*
100 * increment/decrement one channel	2116	1900*
10 * write four channels	395	440*
50 * write four channels	1845	2200*
100 * write four channels	3725	4400*
* - this is an estimation based on no. of commands * clocks required for a single command to execute without FIFO. In reality, this time is bigger due to internal delays (e.g. padring) and datasheet specs		

The above execution times for write commands with FIFO disabled, could be expressed as function of the number of clocks required for an EEPROM read:

$$N_{1,no\_FIFO} = 2 * (eo\_read\_clk\_count + 1) + 7$$
$$N_{2,no\_FIFO} = 4 * (eo\_read\_clk\_count + 1) + 10$$
$$N_{3,no\_FIFO} = 6 * (eo\_read\_clk\_count + 1) + 11$$
$$N_{4,no\_FIFO} = 8 * (eo\_read\_clk\_count + 1) + 12$$
$$N_{40,no\_FIFO} = 80 * (eo\_read\_clk\_count + 1) + 50$$

where $N_{i,\ no\_FIFO}$ is the number of clocks required to execute a write command to i channels. *eo_read_clk_count* is the value programmed in the EEPROM.

Similarly, in case of write commands with FIFO enabled, the expression becomes:

$$N_{i,FIFO} = N_{i,no\_FIFO} + 18$$

To compare the results with the datasheet specs for BUSY low time, a clock period of 20ns, which is the target operating frequency, has been assumed. However, as previously mentioned, there are some other delays added to the right column figures.

Command	BUSY low time	
	Datasheet	Implementation
write one channel	400ns	300ns
write two channels	750ns	520ns
write three channels	1100ns	700ns
write four channels	1450ns	880ns
write all 40 channels	14050ns	7400ns

3.9.3. Calibration Engine

3.9.3.1. Calibration Algorithm

The steps of the calibration algorithm are as follows:

1. x2 = (m + 1) * x1 + c
2. e = x2(LSBs) + start
3. if e < 0: dy = e * (128 LSBs + m1)/(128 LSBs)
   if e >= 0: dy = e * (128 LSBs + m2)/(128 LSBs)
4. x3 = offset + dy

where *start*, *offset*, *m1* and *m2* are calibration coeficients, read from the EEPROM. The format and range of the coeficients is:
- *start[8:0]* - 2's complement, 0.5 LSB precision (range -128 LSBs to 127 LSBs)
- *offset[6:0]* - unsigned, 1 LSB precision (range 0 to 127)
- *m1[7:0]* - 2's complement, 0.25 LSB precision (range -32 LSBs to 31 LSBs)
- *m2[7:0]* - 2's complement, 0.25 LSB precision (range -32 LSBs to 31 LSBs)

They are stored in 2 16-bit words in the EEPROM:
word 0: *start[8:0], offset[6], m1[2:0], m2[2:0]*
word 1: *offset[5:0], m1[7:3], m2[7:3]*

This split of coeficients between the 2 words, especially in case of *m1* and *m2*, has been done to reduce the arithmetic required per clock cycle, as will be detailed later.

Each DAC needs 128 sets of coeficients in the EEPROM, which means 256 16-bit words. The total memory required is 40 * 256 * 16b= 10K * 16b, which is the size of the EEPROM coeficients memory.

To address the coeficients memory, *eeprom_select_coef*=1 and the address is assembled from the DAC address and the MSBs of *x2*:

DAC address						x2[15:9] (MSBs)							0 or 1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

The LSB (bit 0) is 0 for the EEPROM word 0 and 1 for EEPROM word 1.

In the implementation a quarter LSB precision has been used. The implementation details of arithmetic operations are detailed in the following. Because the EEPROM reads are needed right in the middle of the the calibration steps, the arithmetic has been implemented through DesignWare component instantiation for a better scheduling of operations. Also, some operations cannot be inferred, for example multiplication of 2's complement numbers.

1. x2 = (m + 1) * x1 + c

   x2[15:0] = round((m[13:0] + 1) * x1[13:0] + c[13:0])

m and *x1* are unsigned numbers, and *c* is signed represented in 2's complement format. The result is an unsigned number.

This is done in module <calculate_x2> and is strictly not part of the calibration engine.

The first operation is *m[13:0] + 1*. If *m* is full scale, the result overflows, so the half LSB of *m* is discarded and the increment is done as *m_inc[13:0] = m[13:1] + 1*

The result has now a precision of 1 LSB. In Verilog, the DesignWare component instantiation looks like this:

```
wire [13:0] m_inc_wire;
inc1 inc1(.inst_A({1'b0, m_reg[13:1]}), .SUM_inst(m_inc_wire));
```

For the multiplication, a 2-stage pipelined multiplier (unsigned) has been used. This is because it was impossible to fit all the operations (increment, multiplication, addition) in one clock cycle.

```
wire [27:0] m_x1_wire;
mult1 mult1(.inst_A(m_inc_wire), .inst_B(x1_reg), .inst_CLK(clk), .PRODUCT_inst(m_x1_wire));
```

Only the MSBs of the product are kept for addition with *c*, which is padded right with 0's to match the precision of the other operand. A sign position is added to both operands.

```
wire x2_sign;
wire [15:0] x2_wire;
adder2 adder2(.inst_A({1'b0, m_x1_wire[26:11]}), .inst_B({c_reg[13], c_reg, 1'b0, m_x1_wire[10]}),
.SUM_inst({x2_sign, x2_wire[15:0]}));
```

m_x1_wire[10] is included in the second operand because of the requirement to round *x2*:

$$\begin{aligned} & \text{round}(\{1'b0, m\_x1\_wire[26:10]\} + \{c\_reg[13], c\_reg, 3'b000\}) \\ = \; & \{1'b0, m\_x1\_wire[26:11]\} + \{c\_reg[13], c\_reg, 2'b00\} + m\_x1\_wire[10] \\ = \; & \{1'b0, m\_x1\_wire[26:11]\}) + \{c\_reg[13], c\_reg, 1'b0, m\_x1\_wire[10]\} \end{aligned}$$

x2 could underflow/overflow, in which case it has to clamp at zero scale or full scale respectively, so there are flags to detect this:

```
wire x2_uf;
wire x2_of;
assign x2_uf = x2_sign & c_reg[13];
assign x2_of = x2_sign & (~c_reg[13]);
```

2. e = x2(LSBs) + start

   e[10:0] = x2[8:0] + start[8:0]

start is a signed number, and *x2* and unsigned number. The result is signed.

Both *x2* and *start* are extended left with an extra sign position, because the range of the result is wider than the operands (overflow is allowed). *start* is padded right with a 0 to match the precision of *x2*.

e[10:0] = {2'b00, x2_reg[8:0]} + {start_reg[8], start_reg[8:0], 1'b0}
= {{2'b00, x2_reg[8:1]} + {start_reg[8], start_reg[8:0]}, x2_reg[0]}

DesignWare component instantiation looks like this:

```
wire [10:0] e_wire;
adder7 adder7(.inst_A({2'b00, x2_reg[8:1]}), .inst_B({start_reg[8], start_reg[8:0]}), .SUM_inst(e_wire[10:1]));
assign e_wire[0] = x2_reg[0];
```

3. if e < 0: dy = e * (128 LSBs + m1)/(128 LSBs)
if e >= 0: dy = e * (128 LSBs + m2)/(128 LSBs)

dy[10:0] = e[10:0] * (512 + m[7:0]) / 512

(assuming *m* is either *m1* or *m2* depending on the sign of *e*)

e, *m* and *dy* are signed numbers. 512 in straight binary means 128 LSBs represented with a precision of 0.25 LSBs, same as *m*.

This operation has been transformed to speed up the multiplication:

dy[10:0] = e[10:0] * (512 + m[7:0]) / 512 =
= e[10:0] * (10'b1000000000 + {m[7:3], 3'b000}[a] + {2'b00, m[2:0]}[b]) / 512 =
= {e[10:0], 9'b000000000} + {e[10:0] * m[7:3], 3'b000} + {e[10:0] * m[2:0]} =
= ({e[10:0], 9'b000000000} + e[10:0] * m[2:0]) + {e[10:0] * m[7:3], 3'b000}

First step: {e[10:0], 9'b000000000} + {e[10:0] * m[2:0]} =
= {e[10:0], 9'b000000000} + e_m_lsb_temp[13:0] =
= {e[10:0] + e_m_lsb_temp[13:9], e_m_lsb_temp[8:0]} =
= e_m_lsb[19:0][c, d]

Second step: {e[10:0] * m[7:3], 3'b000} =
= e_m_msb[18:0][d]

Final result: dy[10:0] = e[10:0] * (512 + m[7:0]) / 512 =
= (e_m_lsb[19:0] + e_m_msb[18:0]) [18:9][e]

Notes:
[a] - '000' to keep the precision and magnitude of *m* MSBs
[b] - '0' in front of *m[2:0]* is the sign of the *m* LSBs: {m[7:3], 000} + {0, m[2:0]} = m[7:0]. A second 0 is added to use the same multiplier for e[10:0] * m[7:3] and e[10:0] * m[2:0] (11 * 5 2's complement multiplier)

[c] - *e_m_lsb* has been extended with one binary position on the left to account for possible underflows/overflows in the partial sum (these underflows/overflows would disappear ouce the MSB part of the result has been added)
[d] - the last 3 bits can be discarded in *e_m_lsb* because they add with 3'b000 of *e_m_msb* and in the final result they are lost due to division by 512
[e] - bits 8:0 are discarded due to division by 512, and bit 19 has only been used to extend precision for partial result, and is obsolete now

Both multiplications are done using the same multiplier (5 * 11, 2's complement numbers):

```
reg [4:0] mult_A_reg;
wire [15:0] mult_A_B_wire;
mult3 mult3(.inst_A(mult_A_reg), .inst_B(e_reg), .PRODUCT_inst(mult_A_B_wire));
```

Also, the additions in the first step and for the final result are done using the same adder (17-bit numbers):

```
reg [16:0] add_A_reg, add_B_reg;
wire [16:0] add_A_B_wire;
adder4 adder4(.inst_A(add_A_reg), .inst_B(add_B_reg), .SUM_inst(add_A_B_wire));
```

First step:

e_m_lsb[19:0] = {e[10:0] + e_m_lsb_temp[13:9], e_m_lsb_temp[8:0]}

e_m_lsb_temp is the result of the multiplication (mult3) of *e* with

```
mult_A_reg <= (e[10]) ? {2'b00, m1[2:0]}: {2'b00, m2[2:0]};
```

The result of the multiplication is *mult_A_B_wire[15:0]*. The last 3 bits are discarded, bits 8..3 are stored to be used in the final addition, and the rest are added to *e*. Both adder inputs are padded left with the sign because the adder used is wider than needed (it is re-used later).

```
temp_e_m_lsb_reg <= mult_A_B_wire[8:3];
add_A_reg <= {{7{e_reg[9]}}, e_reg};
add_B_reg <= {{10{mult_A_B_wire[15]}}, mult_A_B_wire[15:9]};
```

Second step:

e_m_msb[18:0] = {e[10:0] * m[7:3], 3'b000}

This is straightforward the multiplier output (*mult_A_B_wire[15:0]*), where:

```
mult_A_reg <= (e[10]) ? {m1[7:3]}: {m2[7:3]};
```

The last 3 bits of *e_m_msb* are all 0, and they are ignored.

Final result:

dy[10:0] = (e_m_lsb[19:0] + e_m_msb[18:0]) [18:9]

This is done using adder4 again: *add_A_reg* is *e_m_msb* (the result of the multiplication from second step) and *add_B_reg* is *e_m_lsb* from first step.

```
add_A_reg <= {mult_A_B_wire[15], mult_A_B_wire};
add_B_reg <= {add_A_B_wire[10:0], temp_e_m_lsb_reg};
```

The result of the addition is now *dy*:

dy[10:0] = add_A_B_wire[15:6]

4. x3 = offset + dy

x3[13:0] = round(offset[6:0] + dy[10:0])

offset and *x3* are unsigned numbers, *dy* is signed represented in 2's complement.

The actual operation is

```
x3[13:0] = round(({offset_reg, 9'b000000000} +
            {{6{add_A_B_wire[15]}}, add_A_B_wire[15:6]}) / 4) =
= {offset_reg, 7'b0000000} + {{6{add_A_B_wire[15]}}, add_A_B_wire[15:8]} +
      add_A_B_wire[7]
```

The operation is implemented by a 14-bit adder:

```
adder6 adder6(.inst_A({offset_reg, 7'b0000000}), .inst_B({{6{add_A_B_wire[15]}}, add_A_B_wire[15:8]}),
.inst_CI(add_A_B_wire[7]), .SUM_inst(DAC_data_cal));
```

The result is now the calibrated code for the DAC.

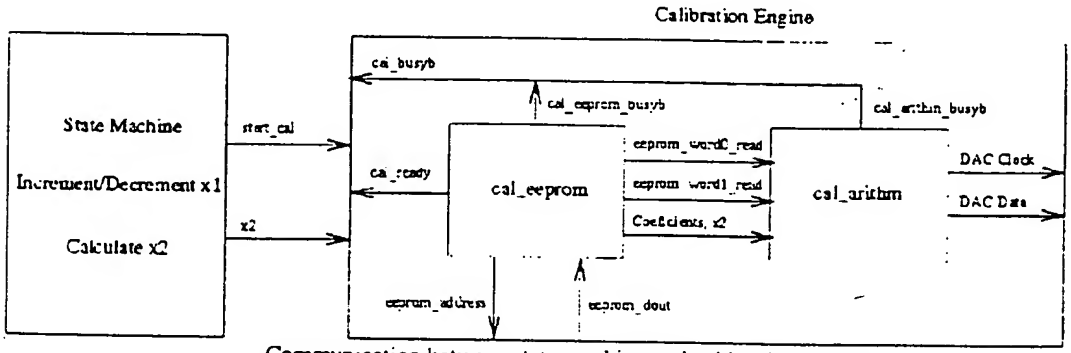
Summary of the arithmetic cells used:

Instance	Type	Implementation	Size
inc1	DW01_inc (increment)	cla	14
mult1	DW02_mult_2_stage (unsigned 2-stage pipelined multiplier)	str	14 * 14
adder2	DW01_add (adder)	bk	17
adder7	DW01_add (adder)	cla	10
mult3	DW02_mult	wall	5 * 11
adder4	DW01_add (adder)	cla	17
adder6	DW01_add (adder)	cla	14

3.9.3.2. |Scheduling

The calibration engine consists of 2 state machines clocked on the rising edge of the same clock. The first state machine (<cal_eeprom>) reads the coeficients from the EEPROM and schedules the multiplications, while the second state machine (<cal_arithm>) schedules the rest of arithmetic operations and generates the signals to write the calibrated code to the DAC. The reason for this implementation was to speed up the execution of a sequence of commands, either read from the FIFO or single commands to write multiple channels.

The 2 state machines communicate between them synchronously and generate signals for communication with the main state machine, as shown in the figure below.

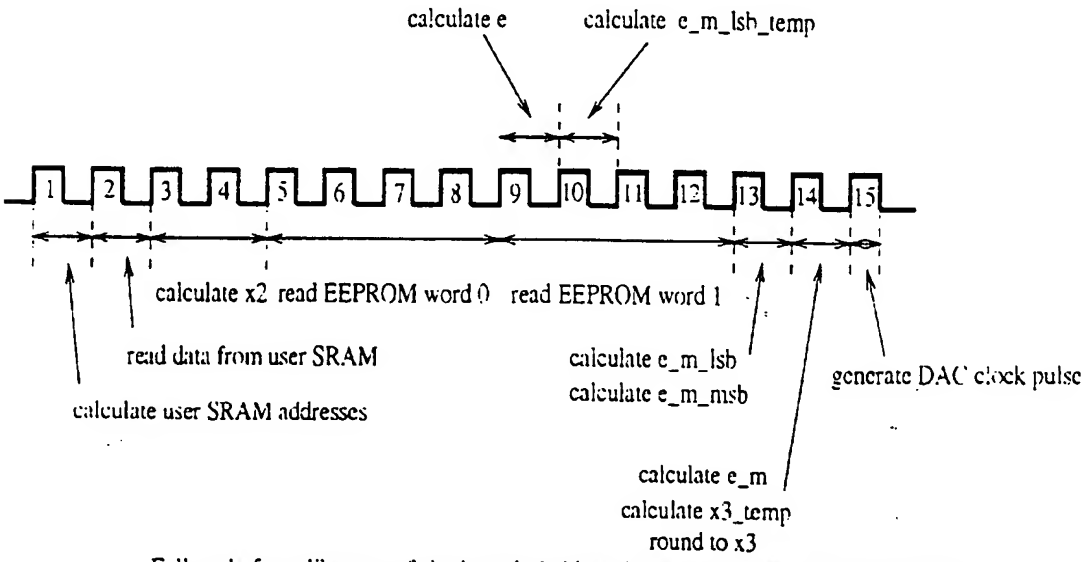


Communication between state machine and calibration engine

The meaning of the signals is as follows:

- *start_cal*: generated by the state machine when the calibration engine is supposed to read the *x2* code and run it through the datapath. The signal stays active (high) and the state machine waits for as long as *cal_ready* is low. If *cal_ready* is high, *start_cal* goes low and the state machine continues with other tasks. The signal also stays high for as long as a multiple channel write command is executed.
- *cal_ready*: generated by the calibration engine when it is ready to take new data. It actually comes from <cal_eeprom> and means that it is ready to read a new set of coefficients from EEPROM. Because the 2 calibration state machines (<cal_eeprom> and <cal_arithm>) are independent, it is possible that *cal_ready*=1 while <cal_arithm> is still executing arithmetic operations.
- *cal_busyb*: activated by the calibration engine while any of the 2 modules are busy. If the state machine has finished executing all the commands, it monitors the status of this signal to decide whether it can go back to sleep mode, or it has to wait a few more clock cycles until the last code is calibrated and written to the DACs.
- *eeprom_word0_read* and *eeprom_word1_read*: used by <cal_eeprom> to signal to <cal_arithm> that the first and second EEPROM words respectively, have been read. Initially, while *eeprom_word0_read* and *eeprom_word1_read* are both low, <cal_arithm> is inactive. When *eeprom_word0_read* goes high, <cal_arithm> starts scheduling the operations which can be done with what was read in the first EEPROM word. After it finishes, loops in a wait state until *eeprom_word1_read*=1. When this happens, it finishes calibrating the code and generate the clock signal for the DACs.

The steps required for the calibration of a single DAC code are shown in the figure below. The notations are those used in section 3.8.5.1.

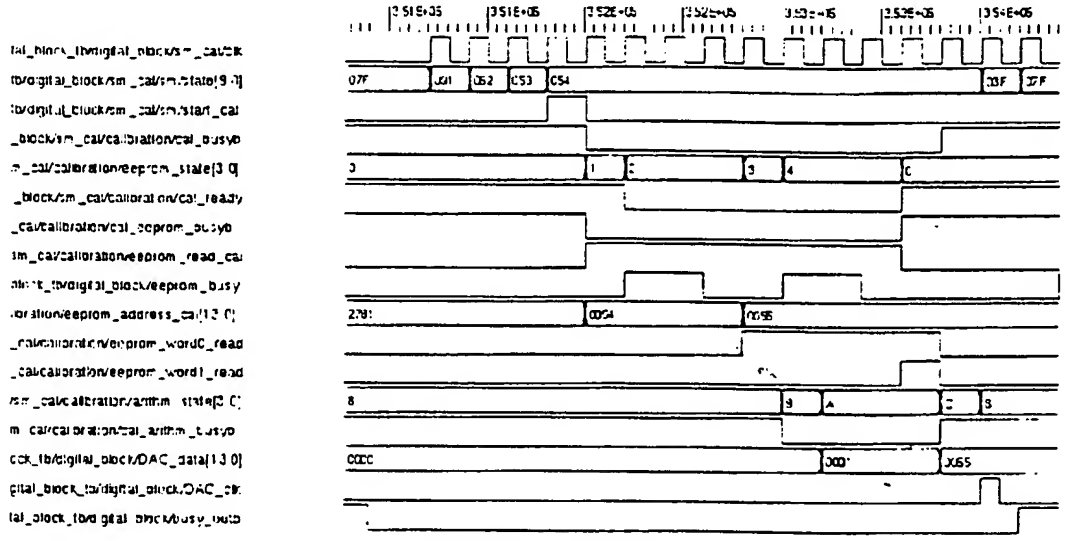


Full cycle for calibration of single code (without increment/decrement and no FIFO)

There are effectively 4 clock cycles to calculate x2 and another 4 to calculate x3, plus one clock cycle to generate the DAC clock. It may also be possible that the 2 operations done while the second EEPROM word is read (calculate e and calculate e_m_lsb_temp) can be merged into one clock cycle, but it was no need for it since the EEPROM read time would be at least 2 clock cycles.

The other 6 clock cycles are spent for reading the EEPROM, so a conclusion is that speeding up the EEPROM read access reduces the execution time.

To exemplify the communication between modules, the following figure shows what happens for calibrating one code.

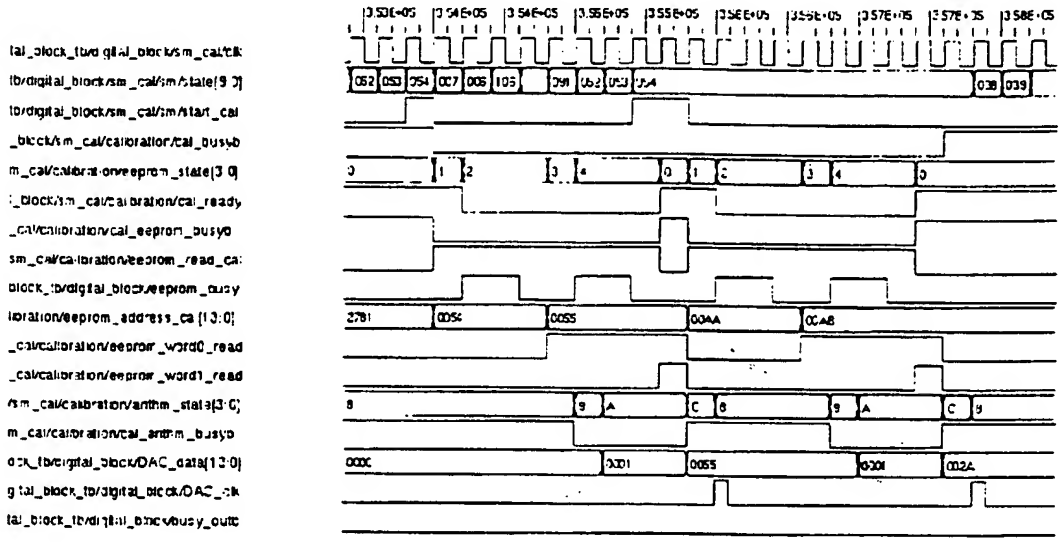


Calibration of one code

First 4 clock cycles are used to calculate *x2*. After the fourth rising edge of the clock, *start_cal*=1, which <cal_eeprom> reads on the fifth rising edge, causing it to move to the next state and load the new EEPROM address. *eeprom_read* and *cal_eeprom_busyb* are both activated. The state machine stays in the wait state (0x054 in the plot) because there is nothing else to execute. <cal_eeprom> goes to the next state and then waits until *eeprom_busy* goes low. When this happens (4 clock cycles after the EEPROM address has been loaded), *eeprom_word0_read*=1 and the new EEPROM address is loaded (only the LSB changes). <cal_arithm> starts because of *eeprom_word0_read* and executes for 2 clock cycles, then stops and waits for *eeprom_word1_read* to activate. 4 clock cycles after the second EEPROM address has been loaded, EEPROM word 1 is read and <cal_eeprom> deactivates *eeprom_read* and *cal_eeprom_busyb* and stops execution. When *eeprom_word1_read*=1, <cal_arithm> resumes execution and outputs the calibrated code and clock signal for the DACs. The state machine, which monitors *cal_busyb*, detects when it goes high and resumes execution to disable the clock generator.

The same applies to all write/increment.decrement commands. An example for 2 succesive commands to write one channel (with FIFO) is shown in the figure below. The FIFO synchronization overhead is not shown for convenience.

The main difference is the behaviour of the state machine: instead of waiting until *cal_busyb* goes high again after the first *x2* was passed, it reads the next commands from the FIFO, decodes it and calculates *x2*. Only then the state machine waits until *cal_busyb*=1. It can be seen that *start_cal* is active high for 2 clock cycles instead of one in the previous example, this being because *cal_ready* was initially 0.



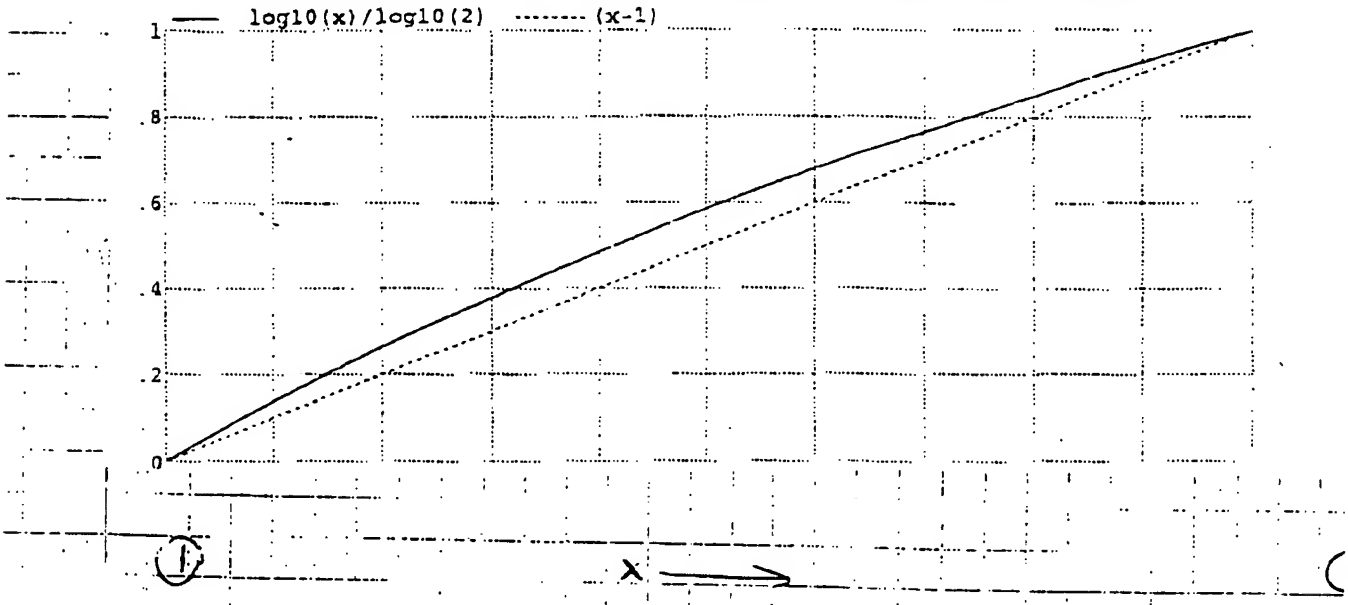
Calibration of two codes

Appendix No. 7:
Engineering Notebook No. 5434,
Pages 10, 12, 13, 19, 20, 21.
Dennis A. Dempsey.
Nov. 6th '00 – Feb. 2001.

6 Nov'00 Calibration Range. Page 10

USER: dempsey Mon Nov 6 16:49:02 2000

Extra Number of bits (Y-axis) vs. Extra Calibration range (X-axis)
Zero reference point (1,0): Fullscale - no extra -> no calibration range
Fullscale point (2,1): Twice the range -> an extra BIT of range.



In order to calibrate via over-ranging & digital calibration, additional range is required. One does not necessarily need full range (an extra BIT) but may suffice with less. Some architectures can make use of this.

eg ADS300

$2^6 + 2^6 - 1 \rightarrow 2^{12}$ bit DAC (4096)

50 + 49 → 2500 steps → 11.2877 bits

40 + 39 → 1600 steps → 10.644 bits

35 + 34 → 1225 → 10.2586 bits

etc

The exact choice of extension of range will depend on the required range and a choice which will be suitable for optimal implementation at a low level.

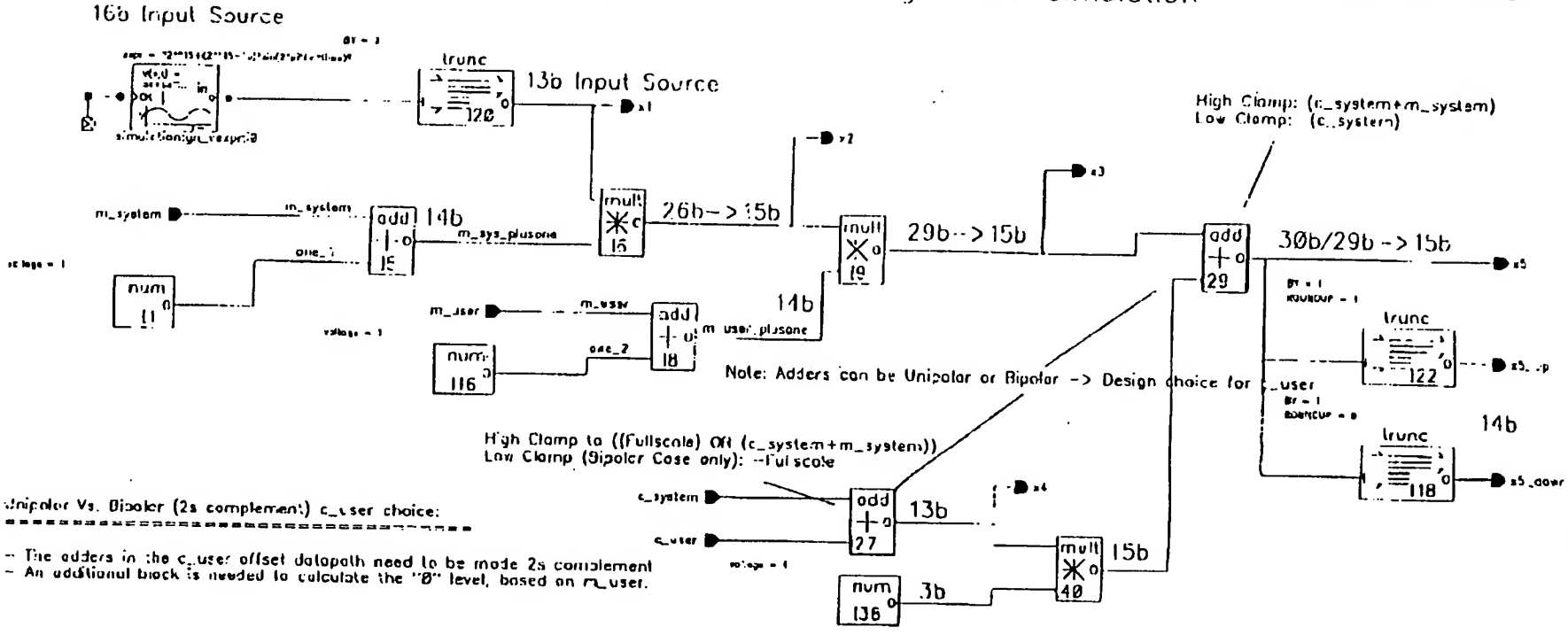
SIGNED Kevin Boerner
WED Kay Goggin
DATE 1/12/2000.
DATE 23/8/2001
SIGNED Denis Dempsey
DATE

- DAC Channel integrated ("System") & user ("_user") offset ("c_") and gain ("m_") calibration.
- System Calibration covers orders 0 & 1 calibration, (usually relatively Large)
- Reduced calibration requirement for DAC calibration (higher order only)

Dennis Dempsey

Jan 25 2001

AD5379 Offset & Gain Calibration High-Level Simulation



DATE 26/1/2001 SUBJECT DAC Calibration 1 of 13 PAGE NO. 12

P12 (across) shows a high level flow description of a DAC calibration methodology to do both user (external) and internal ("system") calibration of gain (order of 1 error) and offset (order of error).

In this embodiment: x (input data) ⇒ 13 bit
m (gain coeff.) ⇒ 13 bit
c (offset coeff.) ⇒ 13 bit

In this embodiment the DAC to be used is 14b resolution. Hence the choice of 15b ⇒ 14b output, which limits the numerically useful input range of x, m & c. e.g. 26b ⇒ 15b for the multiplier I/P

This method of calibration will take out large gain & offset errors. These are most often the dominant accuracy (terribly unadjusted) contributors

This method seperates / isolates the DAC linearity calibration / linearization task from offset & gain.

It allows the DAC Calibration full freedom to make its linearization easier (implementation) and/or cheaper to implement

Also, sharing of hardware is enabled by block choice, again another advantage

WITNESSED AND UNDERSTOOD

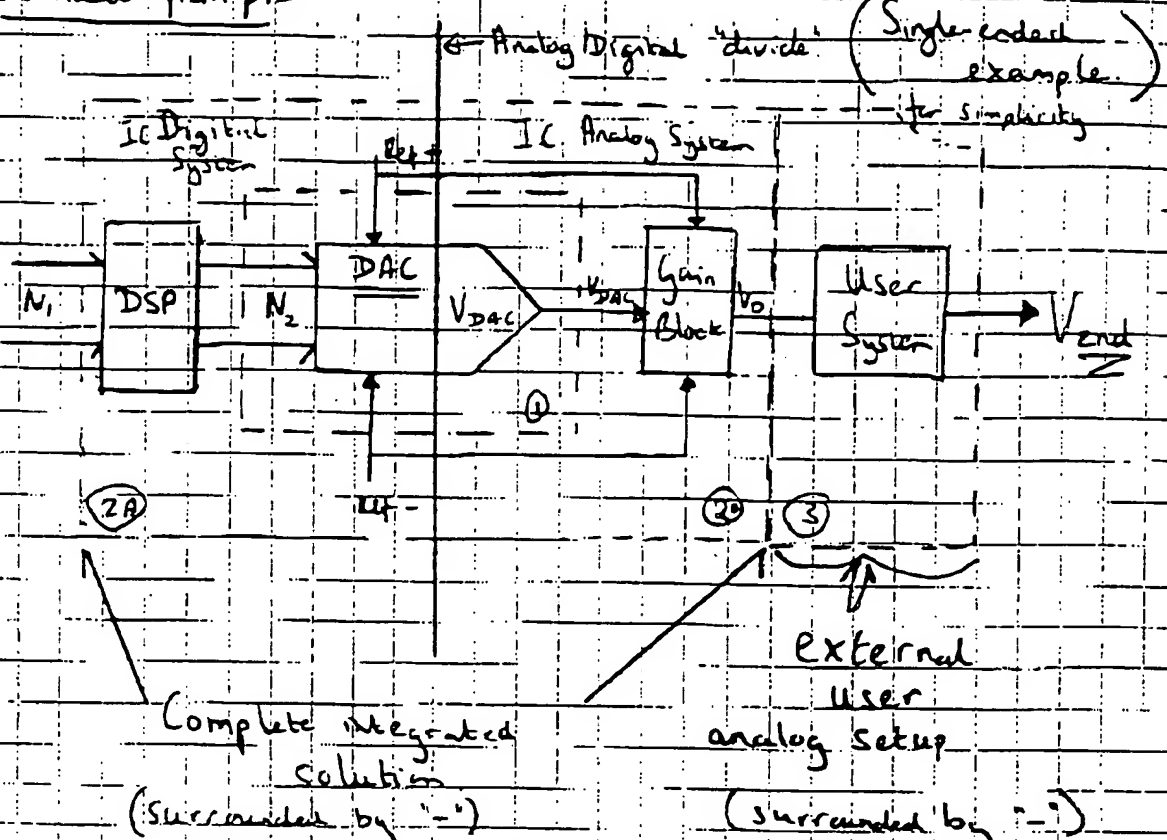
SIGNED [illegible] DATE 26/1/2001

SIGNED Tudor Viorescu DATE 7 February 2001

SIGNED Denis Dempsey DATE 27 Jan 2001

DATE 5 February '01 SUBJECT DAC Solution Calibration PROJECT NO. 1

Continued from p13



① The DAC can be calibrated over its full range, or the useful subset of this. Using the full range always allows a consistent algorithm & implementation to be used for this.

②B Gain can be introduced and/or modulated at the next level in the analog domain. The gain can be over-ranged such that random & non-random manufacturing non-idealities will still allow sufficiently high gain through the channel.

eg ① target gain = 2.0
② gain mismatch = 2.0 ± 10% (6 sigma)
③ Set nominal gain ≥ 2.2 ⇒ 2.23 for 6 sigma design

②A Digitally, we can pre-process the DAC data to reduce the gain through the channel

MESSED AND UNDERSTOOD

VED Tudor Viorescu DATE 7 February 2001 SIGNED Dennis Dempsey

NED [illegible] DATE 20 February 2001 DATE Feb 5. 2001

DATE 5 Feb '01 SUBJECT DAC solution Calibration PROJECT NO.

In the previous example, the nominal analog gain was set = 2.23 although an overall channel gain of 2 is required

∴ Digital gain ⇒ Dig gain * 2.23 = 2
(nominally) ↑ ↑
nominal ideal

Nominal Digital gain = $\frac{2}{2.23}$ = 0.896861...
(Code)
⇒ $\frac{14,694}{16,384}$ (rounded/truncated)

Digital gain Code ⇒ 14,694 @ 14bit level

Per IC, digital gain = $\left(\frac{2}{\text{round}(\text{Actual Gain}} * 16384\right)$

Offsets in the analog section also need to be allowed for by nominally leaving "room" for this. Again, this results in awkward numbers for the DAC "zero" & "full scale" points.

Hence, allowing the gain & offset to be calibrated independently of the higher order non-linear issues has a strong advantage.

→ DAC → one gain, one offset

If you decide to try to integrate a full calibration into the core then larger gain & offset calibration requirements will dominate and therefore require larger numeric range (normally) than converters using the calibration approach offered here.

WITNESSED AND UNDERSTOOD

NED Tudor Viscreanu DATE 7 February 2001 SIGNED Denis Dempsey

NED Oliver Brennan DATE 20 February 2001 DATE February [illegible]

Page 21

DATE 5 Feb 101 SUBJECT DAC Solution Calibration PROJECT NO. 2

As drawn on p19, the user's system is similar to the internal analog gain & offset functions

This means that we can amalgamate our user offset & gain (channel) calibration with an internal per channel one.

Hence user & internal "m" & "c" (gain and offset) calibration are very compatible if a "per channel" offset and gain internal channel calibration is used.

If user calibration is not present, then there is a strange argument to add this to the total DAC calibration

NESSED AND UNDERSTOOD

NED Tudor Kinneen DATE 7 February 2001 SIGNED Denis Dempsey

NED Oliver Brown DATE 20 February 2001 DATE February [illegible]

ANALOG DEVICES
MEMORANDUM

/usr/dsun60/dempsey/dac/frame/
patent3_disc2.doc

D Dempsey
July 2, 1996 3:07 pm

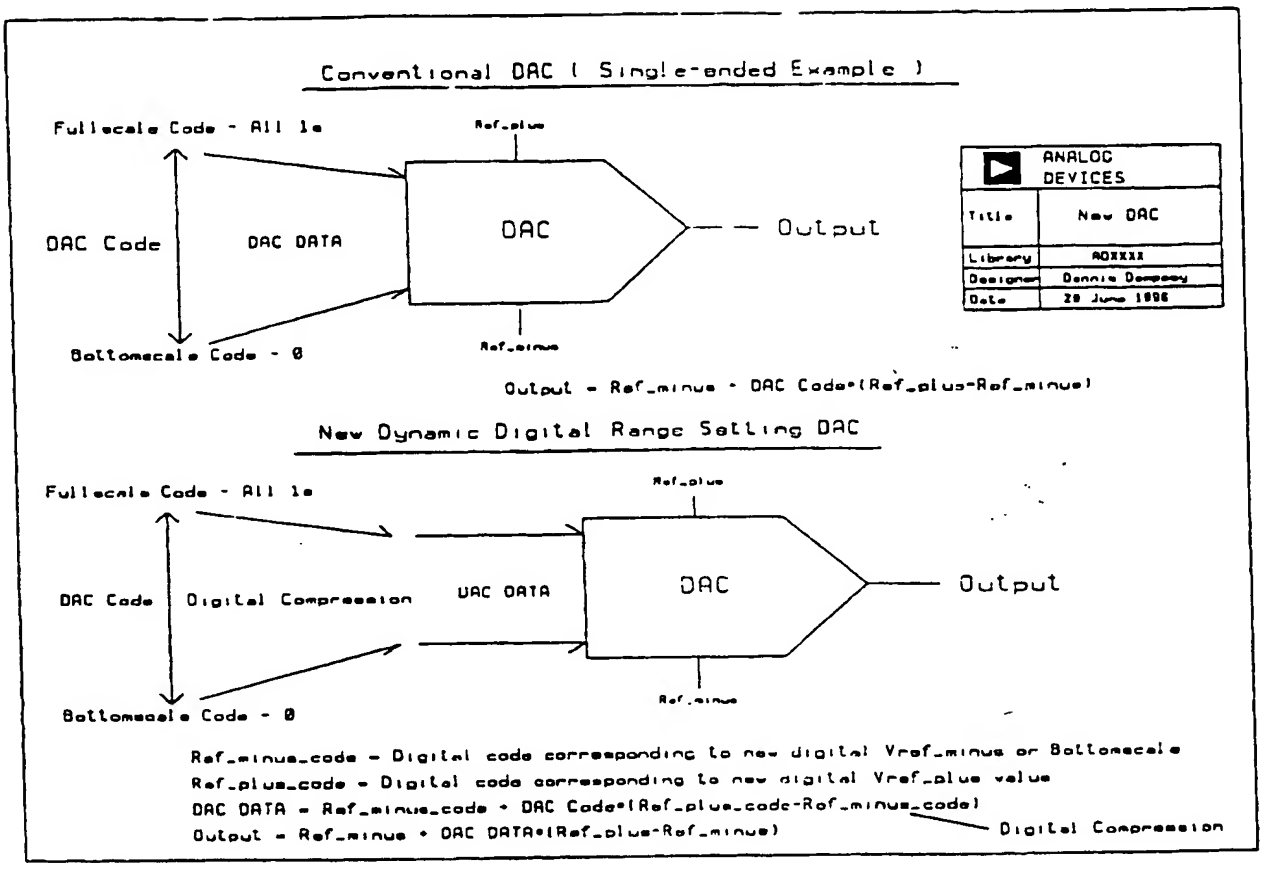


Figure 1: Diagrams of both Convention and New DAC Architecture

ANALOG DEVICES

INVENTION DISCLOSURE

APD1144

Date Submitted: 28 June 1996 (Where Necessary, Use Reverse Side Or Separate Sheet To Complete Answers)

1. Name(s) of Inventor(s): Dennis Arnold Dempsey

2. Subject Matter: New digital data compression approach to maximize spectral performance while giving large user output span control. New DAC approach.

3. Brief Description of Invention: The approach can be used on many DAC architectures. Change is in front of DAC. The DAC DATA is modulated intelligently to give the user-defined output range and offset while optimising the DAC use to maximize spectral performance

4. Detailed Description and Sketches of Invention Are To Be Found on the Attached Sheets, Identified as Pages 4, 5, _____. Forming a Part of This Disclosure.

5. Closest Prior Art Known To Inventor(s). - (Identify by Reference or by Brief Description) Ubiquitous DACs. Do not know of this sort of technique used elsewhere.

6. Closest ADI Practice - (Identify by Spec & Drawing Numbers, or by Brief Description) e.g. DACs AD7834, AD7176,

7. Advantages of Invention Over Art and Practice Identified in 5 and 6 Above
(1) User-defined (digitally) output offset + Span.
(2) Maximized Spectral Performance for the range

Invention Disclosure Page 2

8. Conception of Invention (Date and Record Relied On) 20 December 1995

9. First Written Description (Date and Record, e.g., Lab Notebook No. & Pages)
20 December 1995
Engineering Notebook No. 3854 Pg. 10 & 11.

10. First Drawing or Sketch (Date and Identification)
20 December 1995
Engineering Notebook No 3854 Pg 10 & 11

11. First Disclosure to Others (Date, Person, Record Relied On)
Joe Spalding – 20 Dec 1995
Patrick Griffin – 31 May 1996

12. First Reduction To Practice (Date & Record Relied On) None yet.

13. First Commercial Use (Date & Record Relied On) None yet.

14. Signature(s) of Inventor(s):
Dennis Dempsey (Date) 28 June 1996
(Date)
(Date)

15. Signature(s) of Corroborative Witness(es):
Mark Cotter (Date) 2/July/96
(Date)
(Date)

ANALOG DEVICES
MEMORANDUM

/usr/dsun60/dempsey/dac/frame/
patent3_disc.doc

D Dempsey
June 28, 1996 3:23 pm

1. Field of the Invention:

This invention relates generally to digital-to-analogue converters (DACs).

2. Discussion of the Related Art:

Digital to analogue converters (DACs) generate an analogue output from a digital input word. The output level is related to the input by the following general equation (assumung unity gain):-

Output = Ref_minus + DAC Code * (Ref_plus - Ref_minus) (1)

where : Output = DAC Output, differential or single-ended
Ref_minus = Negative (relative to positive) DAC Output Reference Level input.
Ref_plus = Positive DAC Output Reference Level input
DAC Code = DAC data or code, usually binary.

In order to change the DAC range the user (either external or inside a integrated system) normally changes the reference in order to change the span and offset as per the above equation.

3. Summary of the Invention:

All future DAC Code treatment will assume binary coding for simplicity. Zeroscale output normally produces, as above, the Ref_minus input or some gained up factor of same. Fullscale input produces an output one LSB (least significant bit) less than fullscale or a gained up version of same.

If the user does not want to use the full output range of the DAC then the user can conventionally only use a sub-set of the codes. In order to maximize the spectral use of such codes the user must be very careful and do a lot of 'digital work.'The DAC Code digital input to the DAC can be pre-processed.

A new approach would be to let the user choose to set up a 'zeroscale' and/or 'fullscale' relative to the analog reference inputs which will give a new output range offset and span. This can be done in the analog world with the reference inputs with care but I propose to push this into the digital pre-processing domain. We can have two registers, one for 'digital vref minus' and the other for 'digital vref plus'. One could use a reset to reset these to there conventional levels i.e. zero and fullscale code. If we move away from these values then the output range and span can be modulated digitally. With these new codes the DAC has a lessened code use and therefore reduced signal to noise ratio (SNR), effective number of bits(ENOB) and signal free dynamic range (SFDR). In some applications the user may still want to use their full digital system range and this will happen for user's without modulatable output range. I propose to integrate a datacompression block into the DAC such that the user may still use a normal digital zero to fullscale span but the data compression will 'squeeze' the code span in between the two digitally set up reference levels. If the 'digital vref minus' is set to zero and the 'digital vref plus' is set to fullscale then the DAC can become a conventional DAC !! This is a useful feature.

ANALOG DEVICES

MEMORANDUM

/usr/dsun60/dempsey/dac/frame/
patent3_disc.doc

D Dempsey
June 28, 1996 3:23 pm

The following Figure 1 explains both graphically and in equation form the transfer function of the new DAC. Note the muliplication of the DAC Code and the Reference Code range. This will introduce some error due to the inevitable element of rounding.

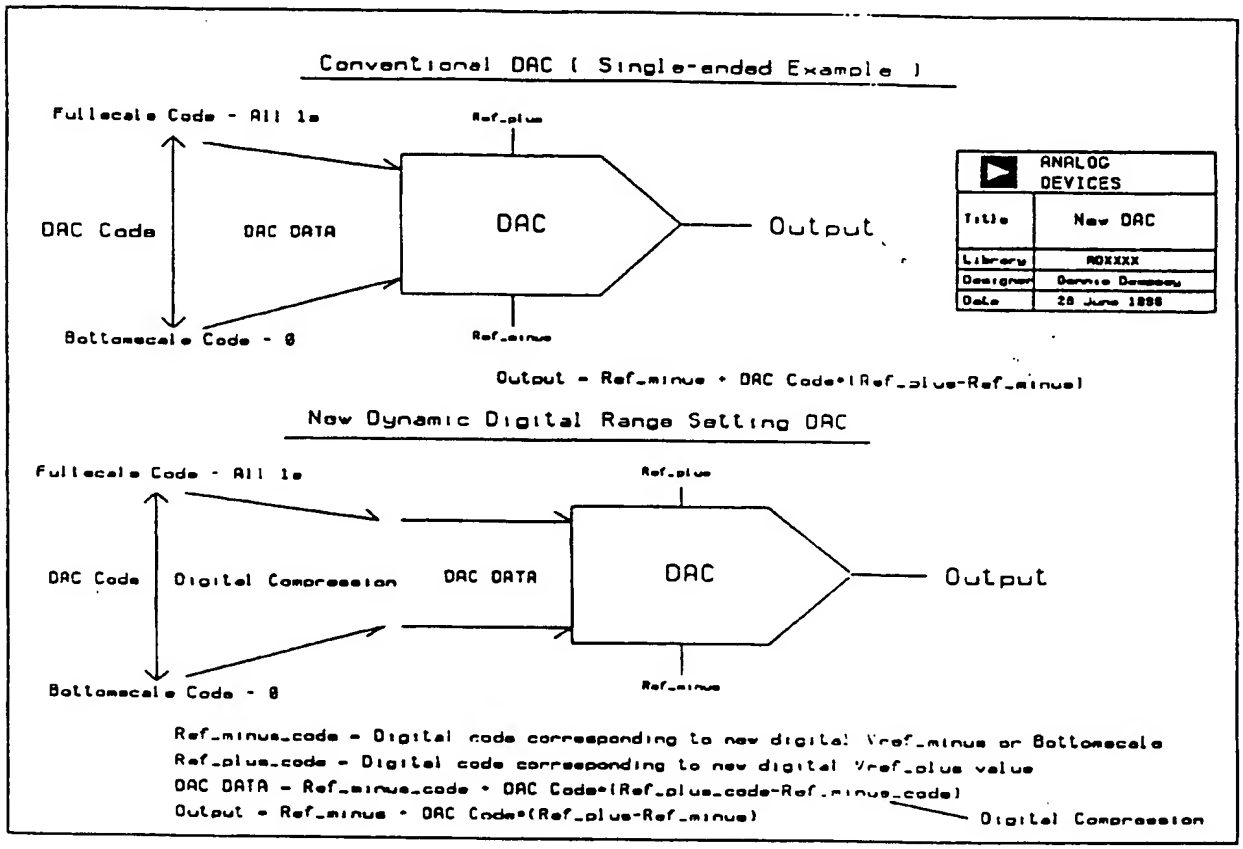
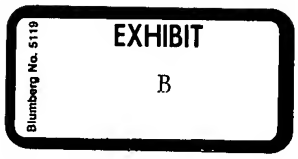


Figure 1: Diagrams of both Convention and New DAC Architecture

Note that allowing the DAC system to use it's full input code range means that the user's SNR is not restricted. The DAC does it's best to get the highest spectral performance. This DAC architecture is very useful in Communications systems and pushing the range control back into the digital domain, rather than analog, generally speaking makes it easier to integrate. It still retains all the flexibility of a conventional DAC.



Appendix No. 4
PrA version of the AD5379 datasheet
(PrA = Preliminary version A, the first one)
Author: Albert O' Grady

40-Channel, 13-Bit, Parallel Input, Voltage-Output DAC

Preliminary Technical Information

AD5379

FEATURES

40-Channel, 13-Bit DACs in One Package
Voltage Outputs
Offset Adjust for Each DAC Pair
References: $V_{REF}(+)$ = +3 V; $V_{REF}(-)$ = -1 V
Output Voltage Range of -2.5 V to +6.5 V
Clear Function to User-Defined Voltage
119-Pin PBGA Package

APPLICATION

Automatic Test Equipment

GENERAL DESCRIPTION

The AD5379 contains forty 13-bit DACs on one monolithic chip. It has output voltages with a full-scale range of -2.5 V to +6.5 V from reference voltages of -1V and +3V

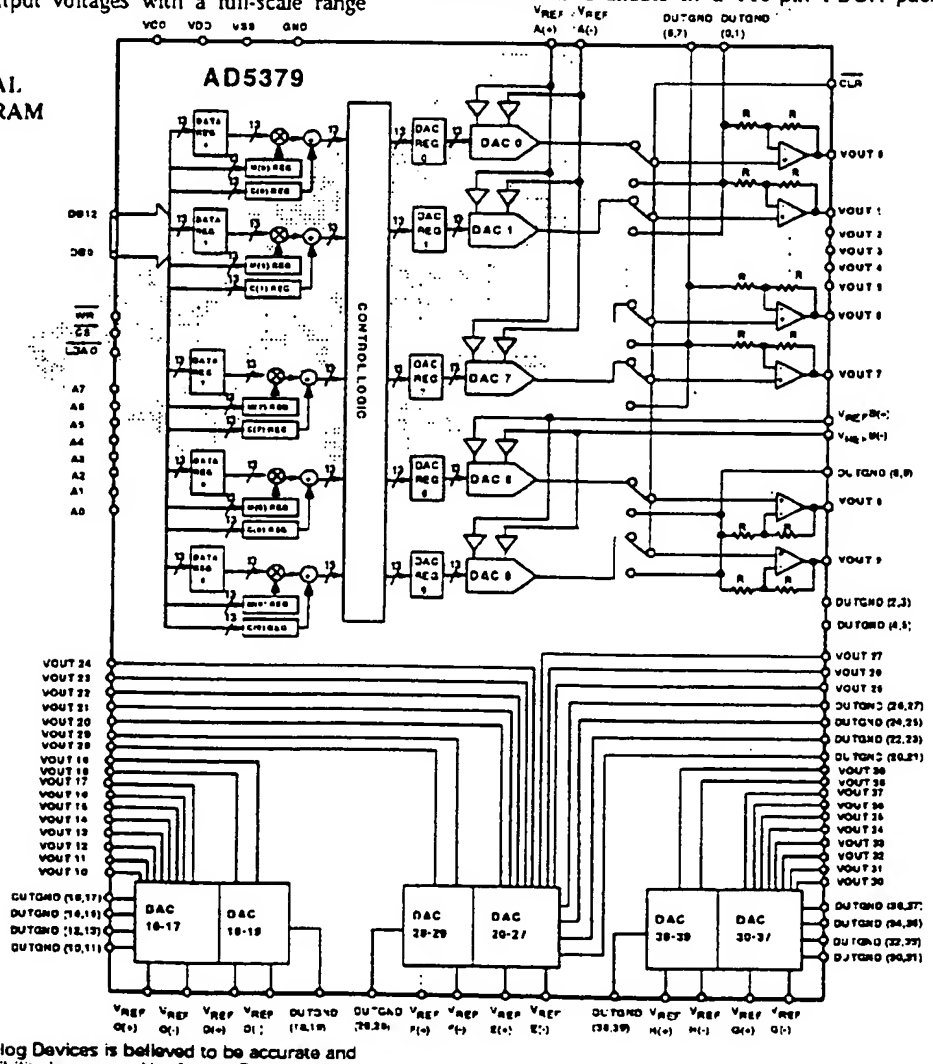
The AD5379 accepts 13-bit parallel loaded data from the external bus into one of the input latches under the control of the $\overline{WR}$, $\overline{CS}$ and DAC channel address pins, A0–A7.

The DAC outputs are updated on reception of new data into the DAC registers. All the outputs can be updated simultaneously by taking the $\overline{LDAC}$ input low.

Each DAC output is buffered with a gain-of-two amplifier into which an external DAC offset voltage can be inserted via the DUTGNDxx pins.

The AD5379 is available in a 119-pin PBGA package.

FUNCTIONAL BLOCK DIAGRAM



REV. PrA Mar. '00

One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A.
Tel: 781/329-4700 World Wide Web Site: http://www.analog.com
Fax: 781/326-8703

AD5379–SPECIFICATIONS

(V_{CC} = +3 V ± 10%; V_{DD} = +9 V ± 10%; V_{SS} = -5 V ± 10% ; V_{REF+} = +3V; V_{REF-} = -1V; GND = DUTGND = 0 V; I_L = 1 mA and C_L = 50 pF to GND, T_A[1] = T_{MIN} to T_{MAX}, unless otherwise noted)

Parameter	AD5379	Units	Test Conditions/Comments
ACCURACY			
Resolution	13	Bits	
Relative Accuracy[2]	±2	LSB max	Typically ±0.5 LSB.
Differential Nonlinearity[2]	±0.9	LSB max	Guaranteed Monotonic Over Temperature. Typically ±0.3 LSB
Zero-Scale Error	±4	LSB max	$V_{REF}(+)$ = +3 V, $V_{REF}(-)$ = -1 V. Typically within ±1 LSB
Full-Scale Error	±4	LSB max	$V_{REF}(+)$ = +3 V, $V_{REF}(-)$ = -1 V. Typically within ±1 LSB
Gain Error	±1	LSB typ	$V_{REF}(-)$ = +3 V, $V_{REF}(-)$ = -1 V
Gain Temperature Coefficient[3]	0.5	ppm FSR/°C typ	
	10	ppm FSR/°C max	
DC Crosstalk[3]	0.25	mV max	
	0.08	mV typ	
REFERENCE INPUTS[3]			
DC Input Resistance	100	MΩ typ	
Input Current	±1	µA max	Per Input. Typically ±30 nA
$V_{REF}(+)$ Range	0/3	V min/max	
$V_{REF}(-)$ Range	-1/0	V min/max	
$\lvert V_{REF}(+)-V_{REF}(-)\rvert$	2/4	V min/max	For Specified Performance. Can go as low as 0V but performance not guaranteed.
DUTGND INPUTS[3]			
DC Input Impedance	60	kΩ typ	
Max Input Current	±1	µA typ	Per Input.
Input Range[4]	±0.5	V min/max	
OUTPUT CHARACTERISTICS[3]			
Output Voltage Swing	-2.5 to +6.5	V min/ max	Output Unloaded.
			V_{OUT} = 2 x ($V_{REF}(-)$+[$V_{REF}(-)$-$V_{REF}(-)$]•D)-V_{DUTGND}
Short Circuit Current	10	mA max.	
Load Current	±1	mA max	To 0 V
Capacitive Load	50	pF max	To 0 V
DC Output Impedance	1	Ω max	
DIGITAL INPUTS			
V_{INH}, Input High Voltage	2.0	V min	V_{CC}=3V±10%
V_{INL}, Input Low Voltage	0.4	V max	
I_{INH}, Input Current[3]	±1	µA max	Total for All Pins. T_A=+25 °C.
	±10	µA max	Total for All Pins. T_A–T_{MIN} to T_{MAX}
C_{IN}, Input Capacitance[3]	10	pF max	
POWER REQUIREMENTS[5]			
V_{CC}	+2.7/+3.3	V min/V max	
V_{DD}	+8.1/+9.9	V min/V max	
V_{SS}	-4.5/-5.5	V min/V max	
Power Supply Sensitivity[3]			
ΔFull Scale/ΔV_{DD}	-90	dB typ	
ΔFull Scale/ΔV_{SS}	-90	dB typ	
I_{CC}	0.5	mA max	V_{INH} = V_{CC}, V_{INL} = GND. Dynamic Current
I_{DD}	40	mA max	Outputs Unloaded. Typically 25 mA
I_{SS}	40	mA max	Outputs Unloaded. Typically 25 mA

NOTES

[1]Temperature range for A Version: 0 °C to +85 °C

[2]Relative Accuracy and Differential Nonlinearity specs are production tested at the conditions above: V_{DD} = +9 V ± 10%, V_{SS} = -5 V ± 10%, V_{REF+} = +3 V, V_{REF-} = -1 V.

[3]Guaranteed by characterization. Not production tested.

[4]See DUTGND Voltage Range (page X)

[5]The AD5379 is functional with power supplies of V_{DD} = +X V and V_{SS} = -X V

Specifications subject to change without notice.

AD5376

AC PERFORMANCE CHARACTERISTICS (These characteristics are included for Design Guidance and are not subject to production testing.)

Parameter	A	Units	Test Conditions/Comments
DYNAMIC PERFORMANCE			
Output Voltage Settling Time	30	µs typ	Full-Scale Change to ±1/2 LSB. DAC Latch Contents Alternately
	50	µs max	Loaded with All 0s and All 1s
Slew Rate	0.7	V/µs typ	
Digital-to-Analog Glitch Impulse	50	nV-s typ	Measured with $V_{REF}(+)$ = +3 V, $V_{REF}(-)$ = -1 V. DAC Latch Alternately Loaded with 0FFF Hex and 1000 Hex. Not Dependent on Load Conditions
Glitch Impulse Peak Amplitude	15	mV max	
Channel-to-Channel Isolation	100	dB typ	See Terminology
DAC-to-DAC Crosstalk	40	nV-s typ	See Terminology
Digital Crosstalk	0.2	nV s typ	Feedthrough to DAC Output Under Test Due to Change in Digital Input Code to Another Converter
Digital Feedthrough	0.1	nV-s typ	Effect of Input Bus Activity on DAC Output Under Test
Output Noise Spectral Density @ 1 kHz	200	nV/(Hz)$^{1/2}$ typ	All 1s Loaded to DAC. $V_{REF}(+) = V_{REF}(-)$ = 0 V

Specifications subject to change without notice.

TIMING SPECIFICATIONS[1] (V_{CC} = +3 V ± 10%; V_{DD} = +9 V ± 10%; V_{SS} = -5 V ± 10 %; GND = DUTGND = 0 V)

Parameter	Limit at T_{MIN}, T_{MAX}	Units	Description
t_1	5	ns min	Address to $\overline{WR}$ Setup Time
t_2	0	ns min	Address to $\overline{WR}$ Hold Time
t_3	20	ns min	$\overline{CS}$ Pulse Width Low
t_4	20	ns min	$\overline{WR}$ Pulse Width Low
t_5	0	ns min	$\overline{CS}$ to $\overline{WR}$ Setup Time
t_6	0	ns min	$\overline{WR}$ to $\overline{CS}$ Hold Time
t_7	4.5	ns min	Data Setup Time
t_8	4.5	ns min	Data Hold Time
t_9	30	µs typ	Settling Time
t_{10}	300	ns max	$\overline{CLR}$ Pulse Activation Time
t_{11}	20	ns min	$\overline{LDAC}$ Pulse Width Low

NOTES
[1]All input signals are specified with $t_r = t_f$ = 5 ns (10% to 90% of 3 V) and timed from a voltage level of 1.2 V.
[2]Rise and fall times should be no longer than 50 ns.
Specifications subject to change without notice.

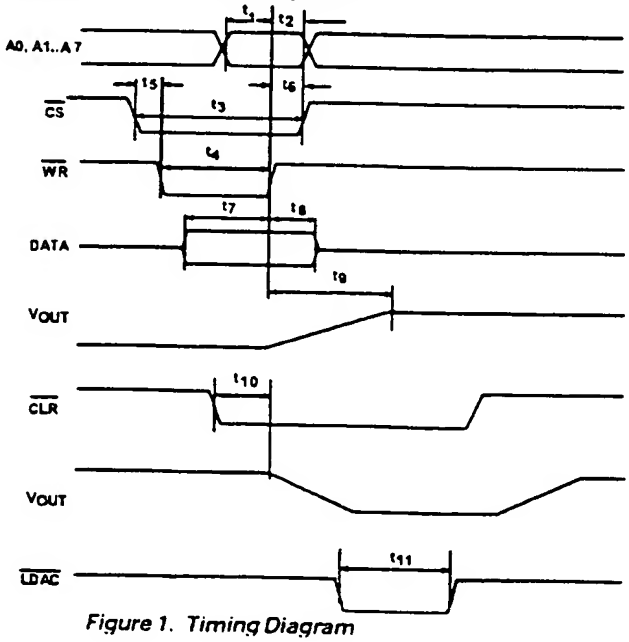


Figure 1. Timing Diagram

AD5376

Address Register Decoding.

The AD5379 contains a nine bit address bus. This address bus provides a number of features as outlined in the tables below. DAC input data registers can be selected independantly, by even address, by odd address or in groups of 4. The address bus also allows each multiply (M) register and each Offset (C) register to be individually updated.

A7	A6	A5	A4	A3	A2	A1	A0

Address Bus

0	0	A5	A4	A3	A2	A1	A0

Update DAC Input Data Register selected by Address A6-A0

0	1	A5	A4	A3	A2	A1	A0

Update Multiply (M) Register selected by Address A6-A0

1	0	A5	A4	A3	A2	A1	A0

Update Offset (C) Register selected by Address A6-A0

1	1	1	1	X	X	X	X

Update all input data registers simultaneously with the same data

1	1	1	0	X	X	X	X

Simultaneously update all input data registers with an Even address

1	1	0	1	X	X	X	X

Simultaneously update all input data registers with an Odd address

1	1	0	0	A3	A2	A1	A0

Simultaneously update a group of input data registers
selected by addresses A3-A0.
A3-A0=0,0,0,0 simultaneously updates DAC registers 0, 10, 20 & 30.

Figure 2. Address Register Decoding Scheme